ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

A THESIS PRESENTED TO THE
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE THESIS
REQUIREMENT FOR THE DEGREE OF THE
MASTERS IN AUTOMATED MANUFACTURING ENGINEERING
M.Ing.

BY
GUILLAUME LATOMBE

FAST INCREMENTAL LEARNING OF STOCHASTIC CONTEXT-FREE GRAMMARS IN RADAR ELECTRONIC SUPPORT

MONTREAL, DECEMBER 12, 2006

THIS THESIS WAS EVALUATED

BY A COMMITTEE COMPOSED BY :

Dr. Éric Granger, Thesis Director
Department of automated manufacturing engineering at École de technologie supérieure

Dr. Jacques-André Landry, Committee President
Department of automated manufacturing engineering at École de technologie supérieure

Dr. Fred A. Dilkes, External Examiner
Defence Research & Development Canada – Ottawa

Dr. Robert Sabourin, Examiner
Department of automated manufacturing engineering at École de technologie supérieure

THIS THESIS WAS DEFENDED IN FRONT OF THE EXAMINATION

COMMITTEE AND THE PUBLIC

ON DECEMBER 11, 2006

AT THE ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

FAST INCREMENTAL LEARNING OF STOCHASTIC CONTEXT-FREE GRAMMARS IN RADAR ELECTRONIC SUPPORT

Guillaume Latombe

ABSTRACT

Radar Electronic Support (ES) involves the passive search for, interception, location, analysis and identification of radiated electromagnetic energy for military purposes. Although Stochastic Context-Free Grammars (SCFGs) appear promising for recognition of radar emitters, and for estimation of their respective level of threat in radar ES systems, the computations associated with well-known techniques for learning their production rule probabilities are very computationally demanding. The most popular methods for this task are the Inside-Outside (IO) algorithm, which maximizes the likelihood of a data set, and the Viterbi Score (VS) algorithm, which maximizes the likelihood of its best parse trees. For each iteration, their time complexity is cubic with the length of sequences in the training set and with the number of non-terminal symbols in the grammar. Since applications of radar ES require timely protection against threats, fast techniques for learning SCFGs probabilities are needed. Moreover, in radar ES applications, new information from a battlefield or other sources often becomes available at different points in time. In order to rapidly reflect changes in operational environments, fast incremental learning of SCFG probabilities is therefore an undisputed asset.

Several techniques have been developed to accelerate the computation of production rules probabilities of SCFGs. In the first part of this thesis, three fast alternatives, called graphical EM (gEM), Tree Scanning (TS) and HOLA, are compared from several perspectives – perplexity, state estimation, ability to detect MFRs, time and memory complexity, and convergence time. Estimation of the average-case and worst-case execution time and storage requirements allows for the assessment of complexity, while computer simulations, performed using radar pulse data, facilitates the assessment of the other performance measures. An experimental protocol has been defined such that the impact on performance of factors like training set size and level of ambiguity of grammars may be observed. In addition, since VS is known to have a lower overall computational cost in practice, VS versions of the original IO-based gEM and TS have also been proposed and compared. Results indicate that both gEM(IO) and TS(IO) provide the same level of accuracy, yet the resource requirements mostly vary as a function of the ambiguity of the grammars. Furthermore, for a similar quality in results, the gEM(VS) and TS(VS) techniques provide significantly lower convergence times and time complexities per iteration in practice than do gEM(IO) and TS(IO). All of these algorithms may provide a greater level of accuracy than HOLA, yet their computational complexity may be orders of magnitude higher. Finally, HOLA

is an on-line technique that naturally allows for incremental learning of production rule probabilities.

In the second part of this thesis, two new incremental versions of gEM, called Incremental gEM (igEM) and On-Line Incremental gEM (oigEM), are proposed and compared to HOLA. They allow for a SCFG to efficiently learn new training sequences incrementally, without retraining from the start an all training data. An experimental protocol has been defined such that the impact on performance of factors like the size of new data blocks for incremental learning, and the level of ambiguity of MFR grammars, may be observed. Results indicate that, unlike HOLA, incremental learning of training data blocks with igEM and oigEM provides the same level of accuracy as learning from all cumulative data from scratch, even for relatively small data blocks. As expected, incremental learning significantly reduces the overall time and memory complexities associated with updating SCFG probabilities. Finally, it appears that while the computational complexity and memory requirements of igEM and oigEM may be greater than that of HOLA, they both provide the higher level of accuracy.

APPRENTISSAGE INCRÉMENTAL RAPIDE DE GRAMMAIRES STOCHASTIQUES À CONTEXTE LIBRE DANS LE SOUTIENT ÉLECTRONIQUE RADAR

Guillaume Latombe

SOMMAIRE

Le Soutient Électronique radar, ou " radar Electronic Support" (ES) en Anglais, comprend la recherche passive, l'interception, la localisation, l'analyse et l'identification d'énergie électromagnétique irradiée à des fins militaires. Bien que les Grammaires Stochastiques à Contexte Libre, ou "Stochastic Context-Free Grammars" (SCFGs) en Anglais, semblent prometteuses pour la reconnaissance d'émetteurs radars, et l'estimation de leur niveau de menace dans les systèmes ES radars, les techniques populaires pour l'apprentissage de leurs règles de production sont très coûteuses en terme de calculs. Les méthodes les plus populaires pour cette tâche sont l'algorithme Inside-Outside (IO), qui maximise la vraisemblance d'un ensemble de données, et l'algorithme Viterbi Score (VS), qui maximise la vraisemblance des meilleurs arbres de dérivation. Pour chaque itération, leur complexité temporelle est cubique par rapport à la taille des séquences de l'ensemble d'entraînement et au nombre de symboles non-terminaux de la grammaire. Comme les applications ES radars nécessitent un protection très rapide contre les menaces, des techniques accélérant l'apprentissage des probabilités de SCFGs sont nécessaires. De plus, dans les applications ES radar, de nouvelles informations du champs de bataille ou d'autres sources sont souvent accessibles à des moments différents. Afin de refléter rapidement les changements dans un environnement opérationel, des techniques incrémentales rapides permettant de mettre à jour les probabilités d'une SCFG seraient un avantage capital.

Plusieurs techniques ont été développées afin d'accélérer le calcul des probabilités des règles de production. Dans la première partie de ce mémoire, trois approches rapides, appelées graphical EM (gEM), Tree Scanning (TS) et HOLA, sont comparées sous plusieurs points de vue – perplexité, estimation des états, capacité à détecter des MFRs, complexité temporelle et mémorielle, et temps de convergence. L'estimation du temps d'exécution moyen, du temps d'exécution extrême et des nécessités de stockage permettent d'évaluer la complexité, alors que des simulations informatiques, exécutées à l'aide de données repréésentant des impulsions radar, permettent d'évaluer les autres mesures de performance. Un protocole expérimental a été défini, de telle manière que l'impact de facteurs tels que la taille de l'ensemble d'entraînement et le degré d'ambiguité puisse être évalué. De plus, puisque VS est connu pour avoir un temps de calcul global plus court en pratique, des variantes VS de gEM et TS sont également proposées et comparées. Les résultats indiquent que gEM(IO) et TS(IO) donnent presque les mêmes résultats en terme de qualité, bien que les ressources requises varient en fonction de l'ambiguité des grammaires.

De plus, bien qu'elles aient des résultats similaires en terme de qualité, les techniques gEM(VS) et TS(VS) procurent un temps de convergence et une complexité temporelle significativement plus faibles en pratique que gEM(IO) et TS(IO). Bien que tous ces algorithmes donnent un meilleur niveau de précision que HOLA, leur complexité de calcul est plus importante. Enfin, HOLA est une technique en ligne qui permet naturellement l'apprentissage incrémental des probabilités.

Dans la seconde partie de ce mémoire, deux nouvelles versions incrémentales de gEM, appelées Incremental gEM (igEM) et On-Line Incremental EM (oigEM), sont proposées et comparées à HOLA. Elles permettent aux SCFGs d'apprendre de nouvelles séquences d'entraînement de façon incrémentale, sans avoir à ré-entraîner depuis le début sur toutes les données d'entraînement. Un protocole expérimental a été défini afin d'observer l'impact sur les performances de facteurs tels que la taille des nouveaux blocs de données pour l'apprentissage incrémental et le degré d'ambiguïté des grammaires de RMFs. Les résultats indiquent que, contrairement à HOLA, l'apprentissage incrémental de blocs de données d'entraînement avec igEM et oigEM procurent le même niveau de précision que l'apprentissage sur toutes les données cumulées depuis le départ, et ce même pour des blocs de données relativement petits. Il a été observé que l'apprentissage incrémental diminue de façon significative les complexités temporelles et mémorielles globales associées à la mise à jour des probabilités des SCFG. Enfin, il apparaît que bien que les complexités temporelle et mémorielle de igEM et oigEM soient plus élevées que HOLA, ils procurent tous les deux un meilleur niveau de précision.

APPRENTISSAGE INCRÉMENTAL RAPIDE DE GRAMMAIRES STOCHASTIQUES À CONTEXTE LIBRE DANS LE SOUTIENT ÉLECTRONIQUE RADAR

Guillaume Latombe

RÉSUMÉ

Modélisation grammaticale pour le Soutien Électronique radar

Le Soutien Électronique, ou "Electronic Support" (ES) en Anglais, radar fait référence à la surveillance passive, l'interception, la localisation, l'analyse et l'identification d'énergie électromagnétique irradiée, à des fins militaires. Le ES procure donc des informations utiles pour la détection de menaces et le déploiement rapide de mesures défensives (Davies et Hollands, 1982; Wiley, 1993). La reconnaissance d'émetteurs radars, et l'estimation instantanée du degré de menace posé par ces radars sont deux fonctions essentielles du ES radar. La récente prolifération et la complexité des signaux électromagnétiques rencontrés dans les environnements modernes ont grandement compliqué ces fonctions.

Dans les systèmes de ES conventionnels, les signaux radars sont généralement reconnus en utilisant les périodicités temporelles à l'intérieur des trains d'impulsions dans des espaces paramétrés, telles que la fréquence porteuse, la fréquence de répétition des impulsions, et la bande passante. Avec l'avènement de la commutation électronique automatique conçue pour optimiser les performances des radars, les radars modernes, et spécialement les radars multi-fonctions, ou "Multi-Function Radars" (MFR) en Anglais, sont souvent bien trop complexes pour être identifiés de cette manière. Les MFRs vont modifier automatiquement, et de façon continue, le signal qu'ils transmettent en réponse aux différents évènements de leurs environnements. Afin d'exploiter la nature dynamique de plusieurs systèmes radars modernes, des algorithmes en traitement du signal basés sur les Grammaires Stochastiques à Contexte Libre, ou "Stochastic Context-Free Grammars" (SCFG) en Anglais, ont été proposés afin de modéliser le comportement des systèmes radars (Visnevski *et al.*, 2005; Visnevski, 2005). De tels modèles peuvent permettre le suivit des comportements dynamiques de modèles d'émetteurs radars, ce qui peut être utilisé pour leur reconnaissance, et pour l'estimation du niveau de menace associé.

Une SCFG G_s est définie par une paire (G, π) où G est une Grammaire à Contexte Libre, ou "Context-Free Grammars" (CFG) en Anglais, c'est à dire une construction mathématique représentée par le quadruplet $G = \{V, N, R, Start\}$. Ce quadruplet consiste en un vocabulaire d'alphabet terminal V, un ensemble de symboles non-terminaux N, un ensemble de règles de production R, et un symbole de départ $Start$. Une règle de production a l'aspect suivant: $\Upsilon \rightarrow \Gamma$, où Υ et Γ sont des éléments de $(V \cup N)^*$ et sont appelés formes sententielles. Le symbole de départ $Start$ est un élément de N. Le lan-

gage formel déterministe Lg généré par une grammaire correspond à l'ensemble des terminaux pouvant être dérivés depuis $Start$ en appliquant R. Le vecteur de probabilités $\pi = (\pi_{A_1}, \pi_{A_2}, ..., \pi_{A_r})$ contient des éléments π_{A_i} qui représentent la distribution de probabilités qu'un non-terminal A_i produise une combinaison de symboles λ. $\theta(A_i \rightarrow \lambda)$ est donc la probabilité que A_i produise λ, et $\pi_{A_i} = (\theta(A_i \rightarrow \lambda), \theta(A_i \rightarrow \mu), ..., \theta(A_i \rightarrow \sigma))$, où $0 \leq \theta(A_i \rightarrow \lambda) \leq 1$ pour λ, et $\sum_\lambda \theta(A_i \rightarrow \lambda) = 1$.

L'apprentissage des grammaires stochastiques

Étant donnés la connaissance *a priori* du comportement d'un radar, et un ensemble de séquences d'entraînement Ω collectées sur le terrain, l'apprentissage des distributions de probabilités associées aux règles de production des grammaires est l'un des défis de l'application pratique des SCFGs. Elles sont généralement estimées selon une méthode d'optimisation à maximum de vraisemblance, ou "maximum likelihood" (ML) en Anglais, en utilisant une technique Espérance-Maximisation, ou "Expectation-Maximization" (EM) en Anglais. Étant donnés une SCFG G_s et un ensemble fini Ω de séquences d'entraînement appartenant à son langage $Lg(G_s)$, la technique EM appliquée à ML pour l'apprentissage des probabilités de la grammaire consiste à maximiser une fonction objective de la forme (Nevado *et al.*, 2000):

$$P(\Omega, \Delta_\Omega | G_s) = \prod_{x \in \Omega} P(x, \Delta_x | G_s) \tag{1}$$

où x est une séquence de Ω, Δ_x est un ensemble donné de dérivations menant à x et Δ_Ω est un ensemble donné de dérivations menant à Ω.

Il peut être noté que cette équation coïncide avec la vraisemblance des séquences d'entraînement (Sanchez et Benedi, 1997) quand Δ_x correspond à l'ensemble $\overline{\Delta}_x$ représentant toutes les dérivations possibles permises par G_s et menant à $x \in \Omega$. On a alors l'égalité: $P(x|G_s) = P(x, \overline{\Delta}_x | G_s)$, qui coïncide également avec la vraisemblance de la meilleure dérivation de la séquence quand Δ_x contient uniquement cette dérivation $\hat{d}_x$ de $x \in \Omega$. Cette maximisation peut être effectuée grâce à l'équation suivante (Nevado et al., 2000; Sanchez et Benedi, 1997):

$$\theta'(A \rightarrow \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s)}{\sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x | G_s)} \tag{2}$$

où $\theta'(A \rightarrow \lambda)$ représente la nouvelle estimation de la probabilité que le non-terminal A émette la combinaison λ.

Les techniques les plus populaires basées sur EM sont les algorithmes *Inside-Outside* (IO) (Baker, 1979; Lari et Young, 1990) et *Viterbi Score* (VS) (Ney, 1992). Alors que IO cherche à maximiser la vraisemblance d'un ensemble d'entraînement, l'algorithme VS (Ney, 1992) cherche à maximiser la vraisemblance des meilleurs arbres de dérivation des séquences de l'ensemble d'entraînement.

Malheureusement, pour des problèmes réels, l'application de IO et VS est limitée à cause de leurs complexités temporelles par itération. En effet, à chaque itération, l'algorithme IO calcule une probabilité interne (*inside*) et externe (*outside*) en deux passages, afin de ré-estimer les probabilités. Lorsque IO estime les probabilités intérieures et extérieures, il passe par toutes les combinaisons possibles de non-terminaux (par exemple $A \rightarrow BC$), comme si n'importe quel non-terminal pouvait produire n'importe quelle paire de non-terminaux, même si, d'après la grammaire, BC ne peut être produit par A. Cela résulte en une complexité temporelle extrême par itération de $O(M_{nt}^3 \cdot L^3)$, où M_{nt} représente le nombre de non-terminaux de la grammaire et L la taille moyenne des séquences. D'un autre côté, il a l'avantage d'avoir une faible complexité mémorielle de $O(L^2 \cdot M_{nt})$.

Contrairement à IO, VS ne nécessite qu'un seul passage pour ré-estimer les probabilités, ce qui résulte en une complexité temporelle par itération plus faible en pratique. Cette technique possède cependant les mêmes complexité temporelle par itération et mémorielle pire cas que IO. Cependant, VS est connu pour converger plus rapidement que IO, bien que les SCFGs ne soient, en général, pas aussi bien entraînées (Sanchez et Benedi, 1999a).

Dans les applications ES radar, de nouvelles informations en provenance du champs de bataille ou d'autres sources sont souvent accessibles à des moments différents. Afin de refléter rapidement les changements dans les environnements opérationels, des techniques incrémentales rapides permettant de mettre à jour les paramètres d'une SCFG seraient un avantage capital. De plus, un apprentissage rapide et incrémental des probabilités permettrait de réduire considérablement les besoins en mémoire et la complexité de calcul associée à la mise à jour des SCFGs pour les applications de ES radar. Or les techniques IO et VS ne permettent pas l'apprentissage efficace de nouvelles informations qui pourraient apparaître lorsque l'environnement opérationnel change. Au contraire, pour apprendre de nouvelles séquences d'entraînement, toutes les anciennes séquences d'entraînement doivent être accumulées et stockées en mémoire, et les probabilités des règles de production doivent être ré-apprises depuis le début sur toutes les séquences.

Ce mémoire présente les résultats d'une exploration d'algorithmes permettant un apprentissage des probabilités d'une SCFG. La première partie du mémoire effectue une étude comparative d'algorithmes rapides pour un apprentissage par lots, tandis que la deuxième partie présente et compare des algorithmes rapides pour l'apprentissage incrémental.

L'apprentissage rapide de SCFGs

Plusieurs alternatives à IO et VS ont été proposées afin d'accélérer l'apprentissage des SCFGs dans différentes applications. Sakakibara *et al.* (1990; 1993; 1994) utilise des grammaires d'arbres, une technique présentant les données de telle façon que l'algorithme évite de passer par toutes les possibilités. Les probabilités sont alors ré-estimées grâce à une généralisation de l'algorithme de Baum-Welch pour les Modèles de Markov Cachés (HMM) et permet d'obtenir une complexité temporelle de $O(L^3 + M_{nt}^3 L)$. Kupiec (1992) utilise la représentation d'une grammaire basée sur un HMM et des diagammes de treillis afin de calculer les probabilités IO. Pour une même grammaire, cet algorithme a la même complexité que IO. Cependant la grammaire utilisée ne doit pas obligatoirement être sous Forme Normale de Chomsky (CNF), ce qui réduit le nombre de non-terminaux M_{nt} et donc réduit la complexité temporelle. Lucke (1994) propose un algorithme BLI (l'auteur ne définit pas cet acronyme) dans lequel les probabilités sont approximées d'une manière également applicable à IO. Il utilise un analyseur syntaxique stochastique et considère l'arbre de dérivation résultat comme une réseau bayesien. Les probabilités sont ré-estimées grâce à deux vecteurs appelés support de noeud causal et support de noeud visible. Ces approximations permettent de réduire la complexité de IO de $O(M_{nt}^3 \cdot L^3)$ à $O(M_{nt}^2 \cdot L^3)$. Ito *et al.* (2001) réduisent la complexité temporelle de $O(M_{nt}^3 \cdot L^3)$ à $O(M_{nt}^2 \cdot L^3)$ en utilisant des grammaires restreintes, dans lesquelles les règles sont de la forme $A \rightarrow AB$ ou $A \rightarrow BA$, où A et B sont des non-terminaux. L'auteur explique que ce type de grammaire peut modéliser de nombreux langages, notamment l'Anglais. Enfin, Chen et Chaudhari (2003) proposent d'utiliser une fonction de prédiction avant d'appliquer IO, afin de détecter des opérations redondantes. Modifier IO de cette façon permet de réduire la complexité temporelle par itération dans une certaine mesure, non spécifiée par les auteurs.

Dans la première partie de ce mémoire, une approche particulière et populaire est considérée pour réduire la complexité temporelle par itération de IO. Elle implique le pré-calcul de structures de données tels que des graphes de support et des histogrammes, à l'aide d'outils comme les analyseurs syntaxiques de Earley (Earley, 1970) ou CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) durant une phase de pré-traitement. Le pré-calcul de structures de données permet alors d'accélérer le processus itératif de ré-estimation des probabilités, puisque les combinaisons aveugles de règles, lorsque n'importe quel non-terminal peut produire n'importe quelle combinaison de non-terminaux, sont évitées. Cependant, ces techniques impliquent nécessairement une complexité mémorielle plus élevée. Cette approche semble plus prometteuse que celles décrites ci-dessus, car elle ne permet pas simplement de réduire la complexité en fonction du nombre de non-terminaux de la grammaire, mais elle dépend de la complexité de la grammaire elle-même. Les grammaires associées aux MFRs étant nettement moins complexes que celles des langages réels, mais appliquées à des séquences plus longues, elle a donc été privilégiée dans ce travail.

Les premiers à exploiter ce genre d'approche sont Fujisaki *et al.* (1989). Leur technique consiste à utiliser une table donnée par un analyseur syntaxique classique, comme les analyseurs syntaxiques de Earley ou CYK, afin de trouver tous les arbres de dérivation possibles produisant une séquence donnée, pour ensuite simplement appliquer l'équation générale de ré-estimation des probabilités. Un algorithme nommé Tree Scanning (TS) est développé dans ce mémoire en s'inspirant du travail de Fujisaki *et al.* Cet algorithme a l'avantage d'être extrêmement rapide une fois que les données ont été pré-traitées, et lorsque la grammaire considérée est peu ambigüe. De plus, les données issues du pré-traitement sont très faciles à stocker, puisqu'elles ne requièrent pas que les règles de production suivent un ordre spécifique. Cependant, les besoins en mémoire deviennent problématiques pour des grammaires assez ambigües. En effet, TS a le désavantage, lorsque l'ambiguïté augmente, d'avoir une complexité temporelle par itération et une complexité mémorielle qui croît avec $O(M_{nt}^{L} \cdot L^3)$ et $O(M_{nt}^{L} \cdot L)$, respectivement. On définit TS(IO) et TS(VS) afin de tester les versions IO et VS de cette technique.

Stolcke (1995) ont proposé un algorithme qui calcule les probabilités internes et externes pendant l'analyse de Earley. Cependant cet algorithme requiert deux passages – un pour le calcul des probabilités internes et un pour le calcul des probabilités externes. De plus, les probablités sont calculées durant l'analyse, et non d'après son résultat, comme c'est le cas pour TS. Cela donne une complexité mémorielle plus faible en pratique pour des grammaires suffisamment ambigües, de $O(M_{nt}^{3} \cdot L^2)$, mais augmente de façon non négligeable la complexité temporelle, qui reste tout de même plus faible que celle de IO et VS en pratique ($O(M_{nt}^{3} \cdot L^2)$), mais l'égale dans le pire cas. Ra et Stockman (1997) introduisent une variante de cette méthode permettant de calculer les probabiliés en un seul passage, grâce à l'utilisation d'un nouveau paramètre. Cette méthode réduit donc de moitié le temps de calcul de l'algorithme de Stolcke, mais augmente sa complexité temporelle dramatiquement à $O(M_{nt}^{6} \cdot L^3)$. Toutefois, la complexité temporelle de cet algorithme est toujours trop élevée pour qu'il soit utilisé dans ce travail.

Sato et Kameya (2001) ont récemment introduit un algorithme appelé *graphical EM* (gEM), qui, comme TS, sépare complètement le processus d'apprentissage EM de l'analyse de l'ensemble d'entraînement. Durant le pré-traitement, gEM crée un ensemble ordonné de graphes de support à partir d'un analyseur syntaxique de Earley ou CYK, afin de ne représenter que les dérivations pouvant mener à une séquence. Pendant le processus itératif, gEM procède d'une manière semblable à l'algorithme IO. La principale différence réside dans le fait que gEM ne passe que par les règles de production décrites dans les graphes de support. Il a été montré que les résultats obtenus avec gEM sont équivalents à ceux obtenus avec IO (Sato et al., 2001; Sato et Kameya, 2001). L'algorithme gEM est plus efficace dans la plupart des cas pratiques, bien que sa complexité temporelle par itération pire cas soit égale à celle de IO. Une complexité mémorielle accrue de $O(M_{nt}^{3} \cdot L^2)$ est cependant nécessaire pour stocker les graphes de support.

Oates et Heeringa (2002) ont présenté un algorithme heuristique en ligne appelé HOLA (les auteurs ne définissent pas cet acronyme) basé sur des statistiques résumées. Contrairement à IO, VS, gEM et TS, HOLA n'optimise pas la vraisemblance de l'ensemble d'entraînement. Cet algorithme ré-estime plutôt les probabilités d'une grammaire en utilisant une approximation de la descente de gradient pour optimiser l'entropie relative entre (a) la distribution des règles de production de l'ensemble d'entraînement (les statistiques résumées) ; (b) la distribution des règles de production d'un ensemble de séquences générées par la grammaire. Dans un premier temps, un pré-traitement est appliqué aux données d'entraînement pour calculer les statistiques résumées: un analyseur syntaxique standard, comme l'analyseur syntaxique de Earley ou CYK, est utilisé pour calculer combien de fois une règle donnée apparait dans la table correspondante. Cette valeur est alors normalisée pour obtenir une distribution sous forme d'un histogramme de référence. La grammaire doit alors générer une certaine quantité de séquences, sur lesquelles le même procédé est appliqué. Les deux histogrammes sont alors comparés et les probabilités sont ajustées en conséquence. L'entropie relative entre les deux distributions est ici utilisée pour évaluer le degré d'apprentissage de la grammaire. HOLA a une complexité temporelle par itération et une complexité mémorielle de $O(M_{nt}^3)$, donc beaucoup plus faible que les techniques précédentes. Par contre le nombre d'itérations peut varier en fonction des différents paramètres, tels que le nombre de séquences générées, la valeur du coefficient d'apprentissage et la distribution initiale des probabilités de production. La faible complexité temporelle et mémorielle de cet algorithme le rend tout à fait adapté pour notre étude.

L'un des objectifs de ce travail est de comparer certaines techniques déjà existantes et adaptées à l'apprentissage rapide de SCFGs pour des applications ES radar. Dans cette optique, les algorithmes TS et gEM d'une part, qui sont tous les deux des approches ML utilisant une technique EM avec leurs avantages et inconvénients respectifs, mais numériquement équivalents, et HOLA, une approche différente optimisant l'entropie relative entre deux distributions grâce à une technique de descente de gradient, ont été sélectionnés. De plus, puisque VS a, en pratique, une complexité temporelle par itération plus faible et converge généralement plus vite que IO, des versions VS de TS et de gEM sont proposées, appelées TS(VS) et gEM(VS), par opposition à TS(IO) et gEM(IO). TS(VS) consiste simplement à calculer les mêmes paramètres que pour TS(IO), mais en appliquant uniquement l'Eq. 2 aux meilleurs arbres de dérivation des séquences de la base d'entraînement, ce qui ne modifie pas les complexités temporelles par itération et mémorielles. gEM(VS) consiste à créer de nouveaux graphes de support pendant le processus itératif à partir des anciens graphes, afin d'accéder à la meilleure dérivation pour une séquence donnée. gEM(VS) requiert donc en pratique une complexité mémorielle plus élevée que gEM(IO). Cependant il a également des complexités temporelles et mémorielles extrême de $O(M_{nt}^3 \cdot L^3)$ et $O(M_{nt}^3 \cdot L^2)$, respectivement.

L'apprentissage incrémental des SCFGs

L'apprentissage des probabilités des règles de production à partir de séquences d'entraînement est particulièrement adapté à des environnements complexes, où une modélisation explicite est difficile. En effet, les systèmes résultants peuvent apprendre et généraliser à partir d'exemples les règles nécessaires à la reconnaissance de MFRs. Cependant, leurs performances dépendent grandement de l'existence de séquences d'entraînement représentatives, et l'acquisition d'un tel ensemble d'entraînement est coûteux en pratique. Les données présentées à un système ES, que ce soit durant l'entraînement ou la phase opérationnelle, peuvent donc être incomplètes. Dans le problème présent, il serait préférable de mettre à jour les probabilités des règles de façon incrémentale, sans corrompre les performances des anciennes données. Dans notre contexte, un algorithme d'apprentissage incrémental doit satisfaire aux critères suivants:

a. il doit être capable d'apprendre des informations additionnelles à partir de nouvelles séquences;

b. il ne doit pas avoir besoin d'accéder aux séquences d'origine utilisées pour l'entraînement de la SCFG existante;

c. il doit conserver les connaissances préalablement acquises.

D'un point de vue incrémental, TS et gEM sont similaires: si de nouvelles données sont ajoutées, le résultat du pré-traitement sur les anciennes données est conservé, et le pré-traitement n'est appliqué que sur les nouvelles données. Le processus itératif doit cependant être appliqué en utilisant les résultats du pré-traitement sur toutes les données. Ces algorithmes sont donc semi-incrémentaux. Leurs complexités temporelles et mémorielles sont raisonnables, voir très faibles dans certains cas, et donc les rendent appropriés pour ce travail. Par contre, HOLA a l'avantage d'être totalement incrémental. Si de nouvelles données sont accessibles, il suffit de mettre à jour l'histogramme de référence, et de reprendre l'entraînement où il avait précédemment été arrêté.

Selon nos résulats antérieurs, HOLA donne des résultats inférieurs à gEM en terme de précision, mais est beaucoup plus rapide. Afin de combiner précision des résultats et rapidité d'exécution, une dérivation incrémentale de gEM serait très pratique dans les applications de ES radar. Deux nouvelles versions incrémentales de gEM – une première approche basique et une seconde permettant de paramétrer la première – ont été proposées dans ce travail, et leurs performances sont caractérisées grâce à une comparaison effectuée avec HOLA.

Dans la littérature, plusieurs variantes de l'algorithme EM peuvent servir d'inspiration pour développer un algorithme gEM incrémental. Rappelons que l'algorithme EM permet l'estimation d'un paramètre θ en optimisant une fonction objective, telle que la vraisemblance de l'ensemble d'entraînement, pour des problèmes dans lesquels certaines variables sont cachées. Une approche de référence pour la ré-estimation de paramètres en utilisant

des données incomplètes a été proposée par Titterington (1984). Cette aproche récursive est adaptée à EM sur un paramètre θ de la façon suivante. Une fonction auxiliaire classique est calculée durant l'étape E, et la récursion est utilisée afin d'approximer la ré-estimation de θ durant l'étape M. À l'itération $k+1$, la récursion calcule θ_{k+1} en utilisant la valeur précédente θ_k, une matrice de Fischer correspondant à l'observation complète des données, et un vecteur de résultats. Jorgensen (1999) a exploré une forme dynamique de EM, basée sur l'algorithme de Titterington, afin de ré-estimer des proportions de mélanges nécessitant une simple mise à jour EM pour chaque observation. Chung et Bohme (2003) ont également amélioré l'algorithme de Titterington en proposant une procédure adaptative afin de déterminer la taille du pas à chaque récursion, ce qui permet d'accélérer le taux de convergence. D'autre part, Neal et Hinton (1999) ont proposé un algorithme nommé "incremental EM", qui permet d'apprendre les paramètres en présentant au système les données de façon séquentielles, dans le but d'améliorer le temps de convergence. S'inspirant de cet algorithme, Gotoh *et al.* (1998) ont présenté une estimation incrémentale des paramètres d'un HMM basée sur EM en sélectionnant un sous-ensemble de données afin de mettre à jour les paramètres, et en ré-itérant le processus jusqu'à convergence. Dans le même ordre d'idée, Digalakis (1999) a proposé une version en ligne de l'algorithm EM qui met à jour les paramètres d'un HMM après chaque séquence de la base d'entraînement, et qui ne nécessite qu'un seul passage au travers des données. Sato et Ishii (2000) proposent une version en ligne de l'algorithme EM appliqué à des réseaux gaussiens. Cette aproche calcule, pour un exemple donné, une approximation de la moyenne pondérée d'une fonction, à l'aide de l'approximation calculée sur l'exemple précédent. Cette approximation permet ensuite de ré-estimer les paramètres du sytème. Enfin, Baldi et Chauvin (1994) ont proposé un algorithme en ligne pour l'apprenissage de HMMs qui n'est pas basé sur EM mais sur une descente de gradient appliquée à la vraisemblance négative. Pour chaque échantillon de données, les paramètres sont ré-estimés en se basant sur l'itération précédente. Les auteurs assurent que cet algorithme peut s'appliquer par lots, de façon séquencielle, ou en ligne.

Il a cependant été décidé de se concentrer uniquement sur les deux algorithmes suivants dans ce travail. Le premier, nommé igEM est inspiré de l'algorithme *incremental EM* de Neal et Hinton (1999), tandis que le second est nommé oigEM et est inspiré de l'algorithme *on-line EM* de Sato et Ishii (2000). Un des principaux avantages de ces deux algorithmes, justifiant leur sélection, réside dans le fait que les optima soient re-questionnés, et que les optima locaux soient possiblement évités. ceci est possible grâce à une réinitialisation des probabilités à chaque ajout de bloc de données, toutes les informations sur les probabilités des règles de production étant stockées dans des vecteurs de statistiques suffisantes. De plus, contrairement à l'algorithme de Titterington, il est facile d'adapter ces algorithmes de manière à se passer de connaître au préalable toutes les données pour la ré-estimation des probabilités. Enfin, oigEM peut en fait être vu comme une version paramétrée de igEM, ces deux techniques étant équivalentes pour une valeur spécifique d'un des paramètres interne de oigEM.

incremental gEM: Neal et Hinton (1998) proposent une version de EM basée sur des vecteurs de statistiques suffisantes au lieu des données brutes, et s'en servent pour définir un algorithme appelé *incremental EM*. Après avoir divisé un ensemble original $\Omega = \{x, Z\}$, où x représente les données observées et Z les données cachées, en sous-ensembles $\{\Omega_1, ..., \Omega_n\} = \{x_1, Z_1, ..., x_n, Z_n\}$, le vecteur de statistiques suffisantes correspondant à chaque sous-ensemble Ω_j est initialisé à une supposition initiale s_j. Un nouveau vecteur est calculé durant l'étape E pour le sous-ensemble considéré uniquement, et θ est alors ré-estimé durant l'étape M en utilisant tous les vecteurs. L'algorithme incrémental de Neal et Hinton met les paramètres à jour en présentant les sous-ensembles Ω_i de façon séquentielle, ou bien en mettant l'accent sur un sous-ensemble pour lequel l'algorithme ne s'est pas encore stabilisé. Cet algorithme n'est donc pas incrémental au sens décrit dans ce mémoire. En effet, à chaque itération, tout l'ensemble de données est requis d'avance, alors que le but de ce travail est de traiter des données non présentes au début de l'apprentissage. Pour cela, une variante nommée *incremental graphical EM* (igEM) est développée ici. On suppose qu'un apprentissage a déjà été effectué sur $\{\Omega_1, ..., \Omega_i\}$ et que un vecteur de statistiques suffisantes résultant s_i est stocké. Si Ω_{i+1} est accessible, on calcule le vecteur de statistiques suffisantes s correspondant durant l'étape E, et on ré-estime θ en utilisant $s + s_i$ durant l'étape M. Une fois que la convergence est atteinte, on stocke $s_{i+1} = s + s_i$, et on recommence jusqu'à Ω_n.

on-line incremental gEM: Sato et Ishii (2000) proposent une version en ligne de l'algorithme EM appliqué à des réseaux gaussiens normalisés. Pour cette application, l'algorithme EM classique est appliqué de la façon suivante: tandis que l'étape E reste la même, l'étape M calcule la moyenne pondérée par la probabilité calculée pendant l'étape E d'une fonction permettant ensuite de ré-estimer les paramètres du réseau. Cet algorithme est dit "en ligne" car les séquences sont présentées individuellement les unes après les autres, en boucle et jusqu'à la convergence, et la moyenne pondérée de la fonction en question est recalculée à chaque nouvel exemple, en se basant sur sa valeur provenant de l'exemple précédent. Afin de rendre cet algorithme incrémental, la fonction considérée est la fréquence d'apparition des règles de production dans les arbres, et les exemples sont présentés en blocs et non plus séparément. La moyenne pondérée de la fréquence des règles de production est calculée en utilisant des vecteurs de statistiques suffisantes. De plus, comme pour igEM, la moyenne pondérée de la fréquence des règles de production est calculée itérativement pour un nouveau bloc d'après le résultat obtenu sur le bloc précédent. Ce nouvel algorithme est appelé *on-line incremental graphical EM* (oigEM).

Ces deux algorithmes sont très semblables, et la principale différence réside dans l'utilisation d'un coefficient d'apprentissage χ pour oigEM permettant de donner plus ou moins d'importance au nouveau bloc de données par rapport aux précédents. Un bref calcul permet de trouver facilement que $\chi(i+1) = \frac{\chi(i)}{1+\chi(i)}$ rend igEM et oigEM équivalents. Cette paramétrisation s'adapte très bien aux applications de ES radar, dans lesquelles on peut

considérer que certaines données sont plus ou moins fiables que d'autres, étant données, par exemple, leurs conditions d'acquisition.

Méthodologie expérimentale

Afin de caractériser les performances des techniques pour l'apprentissage des probabilités des règles de production de SCFGs, une méthodologie expérimentale a été développée. Le modèle hypothétique du système ES radar qui a été utilisé pour les simulations informatiques consiste en un récepteur d'impulsions, un module de reconnaissance de mots, et un module de reconnaissance des séquences.

Pour générer des données, un programme Matlab générant des séquences d'impulsions correspondant à celles de deux MFRs différents – nommés Mercury et Pluto – a été fourni par Recherche et Développement pour la Défense du Canada (RDDC)-Ottawa. Comme un MFR a la capacité de détecter plusieurs cibles en même temps, le programme avait initialement été conçu pour prendre cette fonction en compte. Afin de simplifier le problème, il a été décidé de se concentrer sur une cible uniquement. Les séquences de mots utilisées représentent une détection complète, c'est à dire passe par tous les états possibles du radar – pour Mercury, il s'agit de la Recherche, de l'Acquisition, du Verrouillage non adaptatif, de l'Ajustement de la portée et du Verrouillage continu; pour Pluto, il s'agit de la Recherche, du Verrouillage non adaptatif, de l'Ajustement de la portée et du Verrouillage continu. Afin de représenter la réalité, la durée de chaque état est déterminée grâce à une loi Gaussienne. La durée totale d'une détection et par conséquent le nombre de mots qu'elle contient varient donc d'une séquence à l'autre.

Deux protocoles expérimentaux différents – un pour la première partie de ce mémoire, concernant l'apprentissage par lots, et l'autre pour la seconde partie, concernant l'apprentissage incrémental – ont été définis afin d'évaluer les performances de chaque technique. Dans chaque cas on prend en compte plusieurs facteurs, tels que l'ambiguïté des grammaires (la grammaire de Mercury est plus ambigüe que celle de Pluto), le nombre de séquences de la base d'entraînement pour l'apprentissage par lots, et le nombre de séquences par bloc pour l'apprentissage incrémental. Pour l'apprentissage par lots, un ensemble d'entraînement de 100 séquences est présenté au système et l'apprentissage est effectué dix fois pour chaque taille de la base d'apprentissage, avec des probabilités initiales différentes, jusqu'à convergence de l'algorithme considéré. Afin de mesurer l'impact de la taille de la base d'entraînement sur l'apprentissage, on sélectionne d'abord 25, 50, 75, et enfin les 100 séquences de l'ensemble, comme base d'apprentissage. Le sur-apprentissage est évité grâce à la validation "hold-out" avec une base de validation de taille fixe de 100 séquences, quelle que soit la taille de la base d'apprentissage. Une base de test permet alors de caractériser les performances et de sélectionner le meilleur résultat parmis les dix obtenus. Le test final est effectué en utilisant trois versions d'une autre base de test – une version non bruitée et deux versions avec des niveaux de bruits différents.

Pour l'apprentissage incrémental, un ensemble d'entraînement de 100 séquences est divisé en blocs de 5, 10, 20, 25 ou 50 séquences (afin de mesurer l'impact de la taille des blocks successifs de la base d'entraînement sur l'apprentissage). Pour chaque taille de bloc, l'apprentissage est d'abord effectué dix fois sur le premier bloc, avec des probabilités initiales différentes, jusqu'à convergence de l'algorithme considéré. L'apprentissage est ensuite effectué dix fois sur le second block (de même taille) en intégrant les résultats provenant de l'apprentissage précédent (pour chaque réplication, on associe une réplication provenant de l'apprentissage précédent), jusqu'à ce que toutes les 100 séquences aient été utilisés. Le sur-apprentissage est évité grâce à la validation croisée avec une base de validation de 100 séquences divisée en blocs de 5, 10, 20, 25 ou 50 séquences, comme l'ensemble d'entraînement (à chaque bloc d'entrainement correspond un bloc de validation de même taille). Les test sont effectués de la même façon que pour l'apprentissage par lots.

Étant donné le besoin d'une procédure d'apprentissage qui offre en même temps des résultats précis et une efficacité algorithmique, les performances des différentes techniques sont examinées selon plusieurs perspectives – la perplexité, l'estimation des états, les courbes ROC, le temps de convergence, la complexité temporelle par itération et la complexité mémorielle. Les résultats de nombreuses simulations informatiques sont combinés afin de produire des mesures de performances moyennes.

Résultats

Les résultats, pour la première partie, des apprentissages non incrémentaux indiquent que gEM(IO), gEM(VS), TS(IO) and TS(VS) procurent systématiquement un meilleur niveau de précision que HOLA, mais ont cependant des complexités temporelles et mémorielles significativement plus élevées. À moins que le système MFR considéré soir modélisé par une grammaire très ambigüe, ces techniques sont capables d'apprendre les probabilités rapidement, au prix de ressources mémoire élevées. Les versions VS de gEM et TS ne dégradent pas les résultats par rapport aux versions IO, mais produisent effectivement un temps de convergence et une complexité temporelle par itération significativement plus faibles en pratique. Enfin, le temps d'exécution et les besoins en mémoire de HOLA sont d'un ordre de grandeur plus faible que ceux de gEM et TS. La complexité algorithmique de HOLA est limitée par le nombre de règles de la SCFG, et non par la quantité de données d'entraînement, comme c'est le cas pour gEM et TS.

Les résultats, pour la seconde partie, des apprentissages incrémentaux indiquent que igEM et oigEM donnent de meilleurs résultats que HOLA en terme de précision, mais avec un écart accru par rapport aux versions par lots des algorithmes. De plus, HOLA semble ne pas bien supporter l'apprentissage incrémental, contrairement à igEM et oigEM. Enfin, la possibilité d'apprendre de façon incrémentale permet de diminuer les complexités temporelles et mémorielles de igEM et oigEM, alors que celles de HOLA, étant indépendantes de la taille de la base d'apprentissage, reste inchangées.

Dans une application réelle, la sélection d'une technique particulière par rapport aux autres dépendrait des spécificités de l'application ES, et du rapport désiré entre précision et efficacité algorithmique. Si l'apprentissage se fait par lots, HOLA semble plus adapté au déploiement dans des systèmes de ES radar compacts requérant une protection rapide contre les menaces, alors que gEM et TS semblent plus adaptés aux systèmes de ES radar pour les surveillances à grande échelle, où un temps de réponse plus long peut être toléré afin d'améliorer la précision. Si un apprentissage incrémental est désiré, HOLA ne convient plus, étant donné que les SCFGs résultantes ne sont pas capables de reconnaître les états d'un MFR efficacement, et il devient alors nécessaire d'utiliser igEM ou oigEM.

Quelque soit la technique d'apprentissage utilisée, les grammaires stochastiques nécessitent toujours un temps relativement long lors de leur application sur le terrain. Des expériences ont déjà été menées avec des HMM pour les mêmes tâches. Leur utilisation est plus rapide que celle des grammaires, mais, à cause de leur faible capacité de représentation, ils ne permettent pas de modéliser efficacement les comportements des MFRs. Une troisième alternative se situant entre ces deux premières serait l'utilisation de Réseaux Bayesiens Dynamiques, ou "Dynamic Bayesian Networks" (DBN) en Anglais. Plus puissants que des HMM, mais plus adaptables que des grammaires, ils pourraient permettre d'atteindre le compromis recherché.

ACKNOWLEDGMENTS

First, I want to thank my research director, Éric Granger, who always helped me in my work when I needed, and who pushed me harder that I would have done myself.

I want to thank Fred Dilkes and DRDC-Ottawa for their financial and technical support.

I want to thank all the people from LIVIA who provided me a wonderful ambiance making these years of master very pleasant.

I want to thank all the people with whom I lived during these years passed in Montreal, who were all wonderful people leaving me alone when I needed, and supporting me when I needed.

Most of all, I want to thank my family and friends from France, who supported me even 6,000 kms away.

TABLE OF CONTENT

Page

ABSTRACT i

SOMMAIRE iii

RÉSUMÉ v

ACKNOWLEDGMENTS xvii

TABLE OF CONTENT xviii

LIST OF TABLES xxi

LIST OF FIGURES xxiii

LIST OF ALGORITHMS xxvi

LIST OF ABBREVIATIONS AND NOTATIONS xxviii

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 GRAMMATICAL MODELING IN RADAR ES 7

2.1 Traditional radar ES systems 7
2.2 Challenges of radar ES recognition 9
2.3 Stochastic grammatical modeling of MFRs 11
2.4 Learning production probabilities of SCFGs 19

CHAPTER 3 A SURVEY OF TECHNIQUES FOR FAST LEARNING OF PRODUCTION RULE PROBABILITIES 22

3.1 Classical EM techniques based on ML approximation 23
3.1.1 The Inside-Outside (IO) algorithm 24
3.1.2 The Viterbi Score (VS) algorithm 31
3.2 Fast learning techniques based on ML approximation 36
3.2.1 The Tree Scanning (TS) Technique 39
3.2.2 The Graphical EM algorithm 46
3.2.3 A VS version of gEM 53
3.3 Fast gradient descent based on relative entropy – HOLA 58

3.4 Comparison of learning techniques 66

CHAPTER 4 INCREMENTAL DERIVATIONS OF GRAPHICAL EM 70

4.1 Incremental gEM 73
4.1.1 The Expectation-Maximization algorithm 73
4.1.2 Incremental EM 73
4.1.3 Incremental EM for SCFGs 74
4.1.4 Incremental gEM (igEM) 75
4.2 On-line Incremental gEM 77
4.2.1 EM for Gaussian networks 77
4.2.2 On-line EM 78
4.2.3 On-line gEM for SCFGs 79
4.2.4 On-line incremental gEM (oigEM) 82
4.3 Comparison of igEM and oigEM 85

CHAPTER 5 EXPERIMENTAL METHODOLOGY 89

5.1 MFR radar data 89
5.1.1 The Mercury data 90
5.1.2 The Pluto data 93
5.1.3 The Venus data 95
5.1.4 Languages conversion 96
5.1.5 Ambiguity 97
5.2 Experimental protocol for computer simulations 98
5.2.1 Protocol for batch learning 98
5.2.2 Protocol for incremental learning 99
5.3 Performance measures 101
5.3.1 Log-likelihood (LL) and perplexity (PP) 103
5.3.2 Measures of dispersion 105
5.3.3 Classification rate over radar states 106
5.3.4 ROC curves 107
5.3.5 Convergence time 108
5.3.6 Time complexity per iteration 108
5.3.7 Memory complexity 110

CHAPTER 6 RESULTS AND DISCUSSIONS 111

6.1 Comparison of fast batch learning techniques 111
6.1.1 Quality measures 111
6.1.2 Resources requirements 121
6.2 Comparison of fast incremental learning techniques 126
6.2.1 Quality measures 126

6.2.2 Resources requirements 140

CHAPTER 7 CONCLUSION 144

CHAPTER 8 RECOMMENDATIONS 147

ANNEX 1: Chart parsers 152

ANNEX 2: Non-Implemented Algorithms Based on ML 157

ANNEX 3: Languages and Gramars of MFRs 169

ANNEX 4: Additional results on MFR data 177

ANNEX 5: Experiments on DNA data 180

BIBLIOGRAPHY 193

LIST OF TABLES

Page

Table I — Re-estimation formulas of techniques for batch learning of SCFGs..... 68

Table II — Summary of re-estimation formulas of techniques for incremental learning of SCFGs..... 88

Table III — Conversion tables used to convert the MFRs vocabulary to those of Mercury and Pluto..... 97

Table IV — Confusion matrix associated with PTI, $\texttt{PTN}_1$, and $\texttt{PTN}_2$ for the "best" SCFGs obtained after learning of PTrain with gEM(IO), gEM(VS) and HOLA using the 50 first sequences. 115

Table V — Confusion matrix associated with PTI, $\texttt{PTN}_1$, and $\texttt{PTN}_2$ for the "best" SCFGs obtained after learning of PTrain with gEM(IO), gEM(VS) and HOLA using the first 50 sequences. 116

Table VI — Perplexity of SCFGs obtained with gEM/TS(IO), gEM/TS(VS) and HOLA on MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$. 118

Table VII — Best decision thresholds, and associated HR and FAR, for SCFGs trained with gEM/TS(IO), gEM/TS(VS), and HOLA on MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$. 121

Table VIII — Estimation of the time complexity per iteration for learning techniques. 123

Table IX — Estimation of the memory complexity for learning techniques. 123

Table X — Confusion matrix associated with MTI, $\texttt{MTN}_1$, and $\texttt{MTN}_2$ for SCFGs obtained through incremental learning of MTrain with igEM, oigEM and HOLA, using $|\Omega_i| = 5$. 134

Table XI — Confusion matrix associated with PTI, $\texttt{PTN}_1$, and $\texttt{PTN}_2$ for SCFGs obtained through incremental learning of PTrain with igEM, oigEM and HOLA, using $|\Omega_i| = 5$. 135

Table XII — Perplexity of SCFGs obtained with igEM, oigEM, and HOLA on MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$. 136

Table XIII Best threshold, and associated HR and FAR, for SCFGs obtained with gEM/TS(IO), gEM/TS(VS) and HOLA on MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$ 139

Table XIV Estimates of time complexity per iteration of learning techniques. 141

Table XV Estimates of memory complexity of learning techniques. 141

Table XVI Multiplicative factors for the computation of the overall time and space complexities associated with gEM, igEM and iogEM techniques used for incremental learning. n is the total number of blocks. 142

Table XVII Numerical values of the multiplicative factors for the computation of the overall time and space complexities associated with gEM, igEM and iogEM techniques used for incremental learning. 142

Table XVIII Mercury language. 172

Table XIX Parameters of the states duration for Mercury, in seconds. 173

Table XX Pluto language. 174

Table XXI Parameters of the states duration for Pluto, in seconds. 175

Table XXII VenusA language. 175

Table XXIII VenusB language. 176

LIST OF FIGURES

Page

Figure 1 Block diagram of a traditional radar ES system (Granger, 2002). 7

Figure 2 Example of the TOA of pulses of two different MFRs. 10

Figure 3 Two derivation tree of the sequence "$a\ b\ a\ c\ a$", given the context-free grammar $A \rightarrow A\ b\ A \mid A\ c\ A \mid a$. Here, A is identified with the *Start* symbol, even if it also appears on the right side of production rules..... 14

Figure 4 Block diagram of an ES system that exploits SCFG to model MFR dynamics..... 17

Figure 5 An example of the signal produced for a sequence of pulses that corresponds to a MFR word with fixed PRI, via cross-correlation technique. It represents the probability that the word starts at a TOA, versus the TOA of the pulses..... 18

Figure 6 Branches of a SCFG that are relevant for computation of the inside probabilities of the IO algorithm. 27

Figure 7 Branches of a SCFG that are relevant for computation of the outside probabilities of the IO algorithm. 27

Figure 8 Example of a derivation tree from a short sequence emitted by an MFR... 30

Figure 9 Best derivation tree for example in Fig. 8. 35

Figure 10 CYK tabular chart for example of Fig. 8. 48

Figure 11 Support graphs for example of Fig. 8..... 49

Figure 12 New support graphs for gEM(VS) based on Fig. 11..... 54

Figure 13 Example of derivation tree and sequence generated by the grammar for Mercury..... 63

Figure 14 Illustration of the importance of re-initializing probabilities when training on a new block of sequences..... 87

Figure 15 Pulse envelopes of Mercury words 91

Figure 16 Pulse envelopes of Pluto words and termination character. 93

Figure 17 Data flow diagram representation of the experimental protocol used to evaluate the performance of learning techniques in the first part of this thesis..... 100

Figure 18 Data flow diagram representation of the experimental protocol used to evaluate the performance of learning techniques in the second part of this thesis. 102

Figure 19 Average perplexity (PP) and approximate perplexity ($\widehat{PP}$) of SCFGs trained with for gEM/TS(IO), gEM/TS(VS) and HOLA versus `MTrain` and `PTrain` size. Error bars are standard error of the sample mean. 113

Figure 20 Average convergence time required by the gEM/TS(IO), gEM/TS(VS) and HOLA algorithms, versus `MTrain` and `PTrain` sizes..... 117

Figure 21 ROC curves obtained for SCFGs trained using (a) gEM/TS(IO); (c) gEM/TS(VS); (e) HOLA on `MROCTI`, $\texttt{MROCTN}_1$, and $\texttt{MROCTN}_2$ data, and trained using (b) gEM/TS(IO); (d) gEM/TS(VS); (f) HOLA on `PROCTI`, $\texttt{PROCTN}_1$, and $\texttt{PROCTN}_2$ data. The caption of each subfigure also gives the AUC for `MROCTI`/$\texttt{MROCTN}_1$/$\texttt{MROCTN}_2$ and `PROCTI`/$\texttt{PROCTN}_1$/$\texttt{PROCTN}_2$. 119

Figure 22 Average perplexity (PP) of a SCFG obtained by incremental learning with gEM/TS(IO), gEM/TS(VS) and HOLA on Mercury data for different block sizes. Error bars are standard error of the sample mean. 127

Figure 23 Average perplexity (PP) of a SCFG obtained by incremental learning with gEM/TS(IO), gEM/TS(VS) and HOLA on Pluto data for different block sizes. Error bars are standard error of the sample mean. 128

Figure 24 Average convergence time (I) of a SCFG obtained by incremental learning with igEM, oigEM and HOLA on Mercury data for different block sizes. Error bars are standard error of the sample mean. . 130

Figure 25 Average convergence time (I) of a SCFG obtained by incremental learning with igEM, oigEM and HOLA on Pluto data for different block sizes. Error bars are standard error of the sample mean. 131

Figure 26 Average perplexity for HOLA, igEM and oigEM versus `PTrain` size, for different block sizes. (Error bars are standard error of the sample mean). 137

Figure 27 Average perplexity obtained with igEM versus the training sizes, after training on Mercury data with $|\Omega_i| = 1$ sequence. Incremental learning of blocks is performed with and without re-initializing the probabilities prior to learning each new block of training sequences. 140

Figure 28 Derivation tree of the sequence AaA 168

Figure 29 ROC curves obtained for SCFGs trained using (a) gEM/TS(IO); (b)HOLA on `MROCTI`, $\texttt{MROCTN}_1^*$, and $\texttt{MROCTN}_2^*$. 178

Figure 30 Overall view of experimental protocol for the DNA experiments. 184

Figure 31 Average perplexity SCFGs of gEM(IO) and gEM(VS) versus size of training set, for DATA 10 and DATA 20. (Error bars are standard error of the data set mean.) 187

Figure 32 Convergence time for gEM(IO) and gEM(VS) for DATA 10 and DATA 20 188

Figure 33 ROC curves for the low-ambiguity grammar after training with gEM(IO) and gEM(VS) for DATA-TEST10 and DATA-TEST20 189

Figure 34 ROC curves for the high-ambiguity grammar after training with gEM(IO) and gEM(VS) for DATA-TEST10 and DATA-TEST20 190

Figure 35 ROC curve for the new complex grammar after training with gEM(IO) for DATA-TEST10. 192

LIST OF ALGORITHMS

1 Inside(x) 28

2 Outside(x) 28

3 Inside-Outside() 29

4 Maximum-Probability(x) 33

5 Retrace-path(x) 33

6 Add-store($\psi(i, j|A)$) 33

7 Viterbi-Score() 34

8 Earley-Extract() 39

9 NewTree() 39

10 TreeCompletion() 40

11 CYK-Extract() 40

12 CYK-Trees(A(i,j),numTree) 41

13 Tree-Scanning(IO) 42

14 Tree-Scanning(VS) 43

15 Extract-CYK 47

16 Visit-CYK(l, A, i, j) 47

17 Graphical-EM(IO) 50

18 Get-Inside-Probs() 51

19 Get-Expectations() 51

20 Graphical-EM(VS) 56

21 Get-Inside-Probs-VS() 57

22 Get-Expectations-VS() 57

23 HOLA() 60

24 HOLA-Count() 60

25 HOLA-Iteration() 61

26 Incremental gEM() 76

27 On-line incremental gEM() 83

28 Earley-Parser() 154

29 Earley-Decode() 155

30 CYK-Parser() 156

31 Tree-Scanning(kVS) 159

32 VS-prior-information() 160

33 Tree-Scanning(VS-prior-information) 161

34 RA() 163

35 Compute-Elements() 164

LIST OF ABBREVIATIONS AND NOTATIONS

Brg	bearing
ES	electronic surveillance
EW	electronic warfare
MOP	modulation on pulse
PA	pulse amplitude
PDW	pulse descriptor word
MFR	multi-function radar
PRI	pulse repetition interval
PW	pulse width
RF	radio frequency
TDOA	time difference of arrival
TOA	time of arrival
CNF	Chomsky normal form
CFG	context-free grammar
SCFG	stochastic context-free grammar
CYK	Cocke-Younger-Kasami
x	sequence
Ω	set of sequences

L	length of a sequence
G	a grammar / a context-free grammar
G_s	a stochastic grammar / a stochastic context-free grammar
N	set of nonterminals
V	set of terminals
M_{nt}	number of nonterminals
M_t	number of terminals
r	a production rule
R	set of production rules
$a, b, c, ...$	terminals
$A, B, C, ...$	nonterminals
w_i	i^{th} word of a sequence
$A \rightarrow \lambda$	production rule of a grammar meaning that A is extended to λ
$A \Rightarrow (w_i, ..., w_j)$	notation meaning that A is extended to the subsequence $w_i, ..., w_j$ through several grammar rules
$\theta(A \rightarrow \lambda)$	production probability of rule $A \rightarrow \lambda$
d_x	a derivation tree that forms x
$\hat{d}_x$	best derivation tree that forms x
Δ_x	set of derivations trees that form x
Δ_Ω	set of Δ_x

λ, μ, γ	sequences of terminals and nonterminals
$N(A \rightarrow \lambda, d_x)$	frequency of occurrence of the production rule $A \rightarrow \lambda$ in derivation d_x
$N(A, d_x)$	frequency of appearance of the nonterminal A as a producer of a rule in d_x
$P(), p(), q()$	probabilities
$P(x, d_x \mid G)$	probability that a derivation tree d_x generates x according to G
$P(x, \Delta_x \mid G)$	probability that a set of derivation trees Δ_x generates x according to G
$P(x \mid G)$	probability that a G generates x
$\hat{P}(x \mid G_s)$	probability that a best derivation tree generates x according to G_s
$P(\Omega, \Delta_\Omega \mid G_s)$	probability that a set Δ_Ω generates sample Ω according to G
$P(\Omega \mid G_s)$	probability that a G_s generates Ω
$\alpha(i, j \mid A)$	inside probability that a A generates subsequence $w_i + 1, ..., w_j$
$\beta(i, j \mid A)$	outside probability that a A generates subsequence $w_i + 1, ..., w_j$
$\hat{\alpha}(i, j \mid A)$	maximum inside probability that a A generates subsequence $w_i + 1, ..., w_j$
$\hat{\beta}(i, j \mid A)$	maximum outside probability that a A generates subsequence $w_i + 1, ..., w_j$
$\alpha_{max}(i, j \mid A)$	maximum inside probability that a A generates subsequence $w_i, ..., w_j$
ψ	path of maximum likelihood storage
$\delta_k, \tau_k, \varphi(\tau_k), \nu$	elements of a support graph
$\eta(A \rightarrow \lambda)$	sum over all the strings of the balanced frequency of the rule $A \rightarrow \lambda$

$\hat{\eta}(A \rightarrow \lambda)$	sum on all the strings of the frequency of the rule $A \rightarrow \lambda$ in the best parse trees of the sequences
H	relative entropy for HOLA
HH	total relative entropy for HOLA
χ	parameter for HOLA
LL	log-likelihood
PP	perplexity
T	time complexity per iteration
M	memory complexity
I	convergence time

CHAPTER 1

INTRODUCTION

Electronic Support (ES) involves the passive search for, interception, location, analysis and identification of radiated electromagnetic energy for military purposes. ES thereby provides valuable information for real-time situation awareness, threat detection, threat avoidance, and timely deployment of counter-measures (Davies and Hollands, 1982; Wiley, 1993). Two critical functions of radar ES are the recognition of radar emitters associated with intercepted pulse trains, and the estimation of the instantaneous level of threat posed by these radars. The recent proliferation of complex electromagnetic signals encountered in modern environments is greatly complicating these functions.

In conventional ES systems, radar signals are typically recognized using temporal periodicities within the pulse train in conjunction with histograms of the pulses in some parametric space, *e.g.*, carrier frequency, pulse repetition frequency, and pulse width. With the advent of automatic electronic switching designed to optimize radar performance, modern radars, and especially multi-function radars (MFRs), are often far too complex to be simply recognized in this way. MFRs will continuously and autonomously change their transmitted signals in response to various events in their dynamically-changing environments. In order to exploit the dynamic nature of many modern radar systems, advanced signal processing algorithms based on Stochastic Context Free Grammars (SCFGs) have been proposed for modeling the behavior of radar systems (Visnevski *et al.*, 2005; Visnevski, 2005). Such models can allow tracking of the dynamic behaviour of radar emitter patterns, which can be exploited for recognition of radar emitters, and for estimation of their respective level of threat.

Given some prior knowledge of a radar's behavior, and a set of training sequences collected in the field, one challenge to the practical application of SCFGs is the task of learn-

ing probability distributions associated with the production rules of the grammars. Their estimation is typically performed using an Expectation-Maximization (EM) technique in order to optimize the likelihood of a training dataset. The most popular EM-based techniques are the Inside-Outside (IO) (Baker, 1979; Lari and Young, 1990) and the Viterbi Score (VS) (Ney, 1992) algorithms. The IO algorithm seeks to maximize the likelihood of a training data set, whereas the VS algorithm seeks to maximize the likelihood of the best derivations (parse trees) of a training data set.

Unfortunately, the application of IO and VS to real-world problems is restricted due to the time and memory complexity per iteration and to the large number of iterations needed to converge. Although VS is known to have a lower overall computational cost than IO in practice (*i.e.*, requires fewer iterations to converge, and lower time complexity per iteration), computing the probability, for each iteration, that a SCFG with M_{nt} non-terminal symbols will generate a given string of length L is known has a time complexity of $O(M_{nt}^3 \cdot L^3)$ and a memory complexity of $O(M_{nt} \cdot L^2)$.

Several alternatives to IO and VS have been proposed to accelerate the SCFG learning in different applications (Chen and Chaudhari, 2003) (Ito *et al.*, 2001) (Kupiec, 1992) (Lucke, 1994) (Sakakibara, 1990). A popular approach consists in pre-processing the sequences using standard chart parses before re-estimating the probabilities using the results from parsing. This approach allows to reduce the time complexity per iteration according to the grammar properties, such as complexity and ambiguity. Fujisaki *et al.* (1989) were the first author to adopt this approach. They proposed using a CYK parser to find the derivations or the most probable derivations of the sentences and then directly apply either the Inside-Outside or the Viterbi algorithm. Based on this work, an algorithm called from now on Tree Scanning (TS) (Latombe *et al.*, 2006b) has been introduced, where all the possible derivation trees corresponding to a training set are computed in order to apply the basic reestimation equations. If this algorithm is faster than IO in many practical applications, it has a time complexity of $O(M_{nt}^L \cdot L^3)$, and a memory complexity of $O(M_{nt}^L \cdot L)$ for

a grammar of maximal ambiguity, in which any combination of non-terminals is allowed by the production rules. TS usually corresponds to the case where the most memory is sacrificed to accelerate time complexity.

Stolcke (1995) proposed an algorithm that computes the inside and outside probabilities of IO during an Earley parsing. However, this algorithm requires two passes – one for the inside probability and one for the outside probability. It has the same time complexity per iteration as IO of $O(M_{nt}^3 \cdot L^3)$ in the worst case (when the grammar has maximal ambiguity), that is reduced to $O(M_{nt}^3 \cdot L^2)$ for grammars of limited ambiguity, in which only a given number of combinations of non-terminals are allowed in the production rules, and a memory complexity of $O(M_{nt}^3 \cdot L^2)$. Ra and Stockman (1999) introduced an extension of this last technique that computes both inside and outside probabilities in only one pass, but increases space complexity drastically to $O(M_{nt}^6 \cdot L^2)$. Time complexity then becomes $O(\|r\|^2 L^3)$, where $\|r\|$ is the number of rules of the grammars. However, in the worst case, it is $O(M_{nt}^6 L^3)$, which is greater than that of IO.

Sato and Kameya (2001) initially used a chart parser to produce a special representation of the training data called support graphs, where only the combination of rules leading to the analyzed sequence are represented. Then, during the iterative process they run a new IO-like algorithm based on these support graphs called graphical EM (gEM). Their experiments show a five-fold reduction in time complexity per iteration on the ATR corpus (Uratani *et al.*, 1994). It will be shown in this thesis that its time complexity per iteration and its memory complexity grow with $O(M_{nt}^3 \cdot L^3)$.

Finally, Oates and Heeringa (2002) have introduced a heuristic fully incremental gradient descent algorithm called HOLA based on summary statistics. It uses a standard chart parser to compute the distributions of rules (the summary statistics) found after parsing the training database and after parsing a set of sequences produced by the grammar. Re-estimation of probabilities is performed using an approximation of the gradient descent.

Unlike the other techniques, HOLA does not optimize the likelihood of a sequence (as with IO), but rather the relative entropy between these two distributions. It has the advantage of having the very low time complexity per iteration and memory complexity of $O(M_{nt}^3)$.

In radar ES applications, new information from a battlefield or other sources often becomes available in blocks at different times. Incremental learning of SCFG probabilities, in order to rapidly reflect the changes in the environment, is therefore an undisputed asset. In the present context, incremental learning refers to the ability to update SCFG probabilities from information found in new blocks of training sequences, without requiring access to training sequences used to learn the existing SCFG. Training sequences do not need to be accumulated and stored in memory, and production rule probabilities do not need to be learned from the start on all accumulated data. Furthermore, it should preserve previously acquired knowledge. Incremental learning can considerably reduce the memory requirements and computational complexity associated with updating a SCFG for radar ES applications. Furthermore, since the performance of pattern classification techniques based on the machine learning approach are very dependent on the data used for training, incremental learning permits refinement of a SCFG over time. With the exception of HOLA, the iterative re-estimation process has to start from the beginning, on all cumulative data, to account for new training data. In light of this, an incremental version of gEM would provide a real advantage in radar ES applications.

In literature, several variants of the EM algorithm can provide inspiration for the development of an incremental gEM algorithm. Several EM-based algorithms for on-line or sequential optimization of parameters have been proposed for different applications. They are often recursive (Titterington, 1984) (Jorgensen, 1999) (Chung and Bohme, 2003), sequential (Gotoh *et al.*, 1998), or on-line (Digalakis, 1999) derivations of the Baum-Welsh algorithm (Rabiner, 1989) for re-estimating Hidden Markov Model (HMM) parameters. Some of them are sequential derivations of the general EM algorithm. Other gradient descent techniques for on-line learning can also be found (Baldi and Chauvin, 1994).

Amongst all these techniques, two algorithms were considered for experiments. First, Neal and Hinton (1998) propose an incremental version of the basic EM algorithm, in which blocks of data for training are selected either sequentially, at random, or through a special scheme for which the algorithm has not yet converged. Sato and Ishii (2000) have proposed an on-line version of the EM algorithm for normalized Gaussian networks, in which the examples of the training dataset are presented one after the other until convergence. However, these techniques have yet to be adapted to SCFG learning. It will also be shown that their already existing versions are not incremental in the sense desired in this work, but can be adapted very easily. This is not the case for Titterington's technique and related work, for which a Fischer matrix computed on a complete observation of the data is needed. Moreover, they seem more suitable than gradient descent techniques, that do not usually adapt well to incremental learning because they are likely to provide solutions that get trapped in local optima.

This thesis, divided into two parts, presents the results of a study concerning techniques for fast incremental learning of SCFG probabilities. In the first part, three fast alternatives to the IO technique – gEM (Sato and Kameya, 2001) (named hereafter gEM(IO)), TS(IO), and HOLA (Oates and Heeringa, 2002) – are first compared. Since VS may provide a computational cost that is globally lower in practice, new VS variants of gEM and TS, named gEM(VS) and TS(VS), are also compared. Then, in the second part, two novel incremental derivations of the original gEM algorithm are proposed. The first one, called incremental gEM (igEM), is based on research by Neal and Hinton (1998), whereas the second one, called on-line igEM (oigEM), is based on research by Sato and Ishii (2000). Both algorithms are compared to HOLA.

An experimental protocol has been defined such that the impact on performance of factors like the level of ambiguity of grammars, the size of the training dataset for batch learning, and the size of new data blocks for incremental learning, may be observed. Given the need for a learning procedure that offers both accurate results and computational efficiency, the

performance of these techniques is examined from several perspectives – perplexity, estimation error, convergence time, time complexity, and memory complexity. The data set used in our simulations describes electromagnetic pulses transmitted from a MFR system. The outcome of numerous computer simulations are combined to yield an average performance measure.

The rest of this thesis is structured into six chapters as follows. The next chapter provides some background information on grammatical modeling in the context of radar ES applications. In Chapter 3, the main features of the IO and VS techniques, along with fast alternatives – TS, gEM (both IO and VS versions), and HOLA – are described and contrasted. Chapter 4 describes igEM and oigEM, the new incremental derivations of gEM. Then, the methodology used to compare these techniques, namely, the experimental protocol, data sets, performance measures, is described in Chapter 5 for both parts of this thesis. Finally, the results of computer simulations and complexity estimates are presented and discussed in Chapter 6.

CHAPTER 2

GRAMMATICAL MODELING IN RADAR ES

This chapter first presents the basic functionality of a conventional radar ES system, and the challenge of modern radar ES recognition. Section 2.3 explains how SCFGs can be used to model MFRs. Finally, the fundamental principles involved in learning production probabilities of SCFGs is exposed in Section 2.4.

2.1 Traditional radar ES systems

A radar ES system allows for the passive detection and identification of radar signals for military purpose. As shown in Fig. 1, the basic functionality of current radar ES approaches can be decomposed into three tasks – reception of radar signals, grouping of pulses according to emitter, and recognition of corresponding radar types (Wiley, 1993).

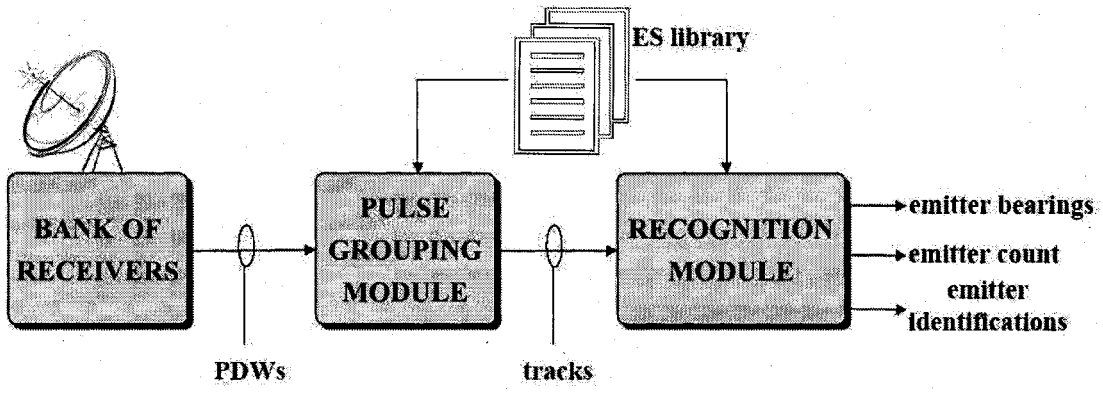


Figure 1 Block diagram of a traditional radar ES system (Granger, 2002).

According to this figure, radar signals are passively intercepted by the receiver portion of the ES system. In typical theaters of operation, intercepted signals are a mixture of electromagnetic pulses transmitted from several emitters. An emitter is an instance of a

radar type, and it is not uncommon to observe several emitters of a same type all being active in a theater of operation. A single type of radar can also operate under several different modes to perform various functions. Simultaneous illumination by these emitters causes overlap and interleaving of the received pulses. Upon detection of a radar pulse, most receivers measure the pulse amplitude (PA), pulse width (PW), radio frequency of the carrier wave (RF) and time-of-arrival (TOA). Direction-finding receivers also measure the bearing (Brg), while advanced receivers also measure the modulation on pulse (MOP). Once parameter values have been measured for a pulse, they are digitized and assembled into a data structure called a Pulse Descriptor Word (PDW).

The stream of successive PDWs is fed to a pulse grouping module, which performs either TOA de-interleaving, or sorting, or both. In short, this module seeks to recover pulse trains and their inter-pulse structure prior to further analysis. This involves progressively grouping pulses that appear to have been transmitted from the same emitter. TOA de-interleaving attempts to discover periodicities in the TOA of pulses using techniques such as TOA difference histogramming (Davies and Hollands, 1982; Wiley, 1993). If periodicities are found, and these correlate with radar intelligence compiled in an ES library, then the corresponding pulses are grouped based on PRI, and stripped away from the input stream of PDWs. Sorting attempts to group pulses based on the similarity of their PDW parameters such as RF, PW and Brg. Gating (Davies and Hollands, 1982; Rogers, 1985) or clustering (Anderberg, 1973) techniques are commonly used to this end.

Recognition makes use of an ES library in which are stored the parametric descriptions of known radar types, and attempts to assign a single radar type to each track. Incidentally, the parametric ranges of various types can overlap in the library, and multiple candidates can appear plausible for the same track, a situation known as an "ambiguity." Therefore, a list of likely radar types is often displayed by an operator interface and monitored over time for every track, along with a confidence rating, threat level, latest bearings, and so

on. Further analysis can assist an ES operator in revealing mode changes in emitters, links between emitters, and inferred platforms.

2.2 Challenges of radar ES recognition

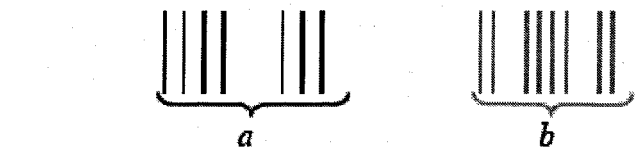
Two critical functions of radar ES are the recognition of radar emitters associated with intercepted pulse train, and estimation of the threat level posed by these radars at any given time. The recent proliferation of complex electromagnetic signals encountered in modern environments is greatly complicating these functions. In order to perform these functions, ES systems must keep evolving in response to the agility radar signals, and to power management and low probability of intercept waveforms.

The multiplication of radar modes is the result of computer control and the ease with which parameters such as RF and PRI can be changed. From an ES standpoint, agility in these parameters can make pulse grouping very difficult, and ES libraries very complex. It is difficult and expensive to maintain comprehensive ES libraries that accurately reflect each specific operational environment. Library construction requires explicit modeling of known radar systems, based on prior information and data. This task is complex, tedious, and prone to error. Owing to the multiplication of modes, it is not uncommon for a library to be incomplete and to contain erroneous data. A shorter response time requires faster pulse grouping, as well as recognition using fewer pulses. In addition, the occurrence of low power waveforms implies that pulses near the receiver detection threshold may be dropped, and hence that pulse grouping must work satisfactorily on sparse data. Finally, response time is critical if threats are to be avoided, or self-protection measures such as chaff dispensing, maneuvering, or electronic jamming, are to be successful.

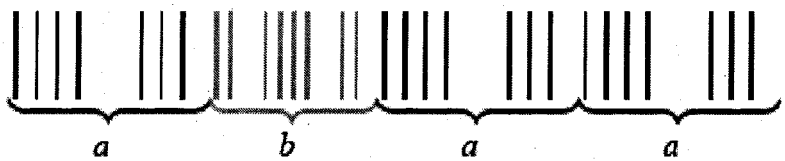
In conventional ES systems, radar signals are often recognized using temporal periodicities within the pulse train in conjunction with histograms of the pulses in some parametric space, e.g., frequency and pulse width. These approaches are ill-suited to exploit the fact that many modern radar systems are highly dynamic and can frequently change their trans-

mitted signals in response to various events. A drawback of histogramming of parameters associated with individual pulses is that most of the temporal relationships amongst the pulses is typically lost. On the other hand, the limitation of periodic temporal analysis is that it assumes that the radar system is a stationary source of pulses. This holds true only for very simple radar systems, and often only over short periods of time.

With the advent of automatic electronic switching designed to optimize radar performance, modern radars, and especially multi-function radars (MFR), are usually far too complex to be recognized using temporal periodicities within the pulse train. MFR will continuously and autonomously switch from one type of signal to another to adapt to the changing environment. Such changes can occur, for example, when the radar detects or abandons targets and consequently switches amongst its search, acquisition and tracking functions, or when a missile is engaged and requires command guidance. The radar emitter is partially driven by the target. Moreover, some electronically steered radar systems may perform many functions simultaneously, greatly increasing the complexity of the radiated signal. Track-While-Scan (TWS) radars and Multi-Function Radars (MFRs) can for instance simultaneously engage multiple targets.



(a) Two MFR words of pulses and identified with symbols a and b.



(b) Sequences of symbols represent concatenated radar words.

Figure 2 Example of the TOA of pulses of two different MFRs.

In light of current challenges in radar ES, more powerful approaches are sought to achieve enhanced accuracy and reliability for radar type recognition, and for instantaneous estimation of threat levels.

2.3 Stochastic grammatical modeling of MFRs

Haykin and Currie (2003) and Lavoie (2001) attempted to apply *Hidden Markov Models* (HMM) to the problem of emitter recognition and threat evaluation. HMMs are a statistical framework for modeling systems that follow a Markov process. It can be defined as a stochastic model in which only observable states are accessible, and whose purpose is to determine the hidden states. In their model, the observable states of the HMM would correspond to time-windowed observations of pulses, while the corresponding radar state (`Search`, `Acquisition`, *etc.*) would represent the hidden states. They concluded that basic HMMs were not suitable for modeling MFR systems when used as the only processor, because of the complexity of the MFRs and their adaptive behaviours to the environment. They added that a hierarchical structure is needed in order to perform the different tasks of an emitter recognition system, consisting in a word recognizer, a sequence recognizer, and a state estimator.

A HMM can be seen as a particular simple type of grammar called *Regular Grammars* (RG). Using a *Context-Free Grammar* (CFG) – a more complex class of grammars – may provide an efficient means to model MFR systems. In particular, signal processing algorithms based on *Stochastic Context-Free Grammars* (SCFGs) constitute one promising approach (Dilkes, 2005a; Visnevski *et al.*, 2003) for future ES systems. The rest of this section provides some background information on modeling of MFR with deterministic and stochastic context-free grammars.

In modern radar ES applications, pulsed radar signals are generated by a MFR in reaction to its current operating environment. For instance, when a radar detects or abandons targets it switches among its `Search`, `Acquisition` and `Tracking` functions – also named

radar states. The algorithm controlling the function of a MFR is designed according to stochastic automata principles, and the state transitions within the automata are driven by the stochastic behavior of the targets (Visnevski, 2005). Consequently, MFR have a finite set of behaviors, but the transition sequence among them is unpredictable. The resulting signals from MFRs may be decomposed into two levels of data organization – the *pulse level*, and the *word level*. Radar *words* can be defined as certain static or dynamically-varying groups of pulses that a MFR emits in different states, as shown in Fig. 2(a). In addition, a concatenated sequence of several words may form a *phrase*, which corresponds to a state of the radar. The number of words per phrase, their structure, etc., varies according to the MFR.

A deterministic formal *language* Lg is defined to be a set of finite sequences of symbols drawn from some finite vocabulary V. Linguistic modeling of a radar system's behaviour may be achieved if one identifies symbols of the vocabulary with the words of a specific MFR, as illustrated in Fig. 2(a). By concatenating the corresponding words together, as shown in Fig. 2(b), a language may represent all possible sequences of words that a radar could ever emit, from power-up to shutdown. For electronically agile radar systems, the language can be quite sophisticated and does not have a straightforward description. However, one can create a finite set of grammatical rules to describe a particular language associated with complex radar systems. (Visnevski *et al.*, 2005; Visnevski, 2005).

A *grammar* G is a mathematical construction represented by the quadruplet $G = \{V, N, R, Start\}$. It consists of a vocabulary or terminal alphabet V, a set of nonterminals symbols N, a set of production rules R, and a start symbol $Start$. A production rule has the following aspects: $\Upsilon \rightarrow \Gamma$, where Υ and Γ are elements of $(V \cup N)^*$ – which means that they are combinations of undefined length of elements of V and N – and are called sentential forms. The start symbol $Start$ is an element of N. There is a unique empty string represented by ϵ, which is an element of V^*.

It is possible to classify grammars according to one of the four following families in the Chomsky hierarchy (Fu, 1982), as defined by the form of their production rules (in this description, an upper-case letter is an element of N, a lower-case letter an element of V, and a Greek letter is an element of $(V \cup N)^*$):

a. the regular grammars (RG): $A \rightarrow aB$, or $A \rightarrow a$;

b. the context-free grammars (CFG): $A \rightarrow \lambda$;

c. the context-sensitive grammars (CSG): $\Sigma_1 A \Sigma_1 \rightarrow \Sigma_1 \lambda \Sigma_2$, where Σ_1 and $\Sigma_2 \in (V \cup N)^*$ and $\lambda \in (V \cup N)^* \setminus \{\epsilon\}$;

d. the unrestricted grammars (UG): not defined by any specific rule.

Thus, the Chomsky hierarchy can be summarized by: RG $\subset$ CFG $\subset$ CSG $\subset$ UG. Consider a *Context-Free Grammar* (CFG) G, corresponding to the four-tuple $\{V, N, R, Start\}$, where $N = \{Start, A, B, ..., C\}$ is a finite set of non-terminal symbols, $V = \{a, b, ..., c\}$ is a finite set of terminal symbols ($V \cap N = \emptyset$), $Start \in N$ is the initial non-terminal symbol, and R is a finite set of rules of the form $A \rightarrow \lambda$ where $A \in N$, $\lambda \in (V \cup N)^*$. Only grammars with no empty rules are considered here.

A *derivation tree* d_x, of a sequence $x \in V^*$ in G, is a sequence of rules $(r_x^1, r_x^2, ..., r_x^m) = d_x$, $m \geq 1$, such that the x is generated from the $Start$ symbol, by successively generating combinations of terminals and non-terminals $\Sigma_i \in (V \cup N)^*$: $(Start \overset{r_x^1}{\Rightarrow} \Sigma_1 \overset{r_x^2}{\Rightarrow} \Sigma_2 \overset{r_x^3}{\Rightarrow} ... \overset{r_x^m}{\Rightarrow} x)$. The *language* generated by G is defined as $Lg(G) = \{x \in V^* | Start \Rightarrow x\}$, that is the set of terminals that can be derived from $Start$ by applying the production rules in R – it can also be seen as a particular subset of V^*. In the example of Fig. 3 (a), $d_x = (r_x^1 \equiv A \rightarrow AcA, r_x^2 \equiv A \rightarrow AbA, r_x^3 \equiv A \rightarrow a, r_x^4 \equiv A \rightarrow a, r_x^5 \equiv A \rightarrow a)$, that gives $(A \overset{r_x^1}{\Rightarrow} AcA \overset{r_x^2}{\Rightarrow} AbAcA \overset{r_x^3}{\Rightarrow} abAcA \overset{r_x^4}{\Rightarrow} abacA \overset{r_x^5}{\Rightarrow} abaca)$(note that the production rules were applied from left to right on the intermediary sequences of symbols, otherwise

the same set of production rules may lead to another sequence of terminal symbols). A CFG is said to be *unambiguous*, if for each $x \in Lg(G)$, there exists only one derivation; otherwise it is called ambiguous.

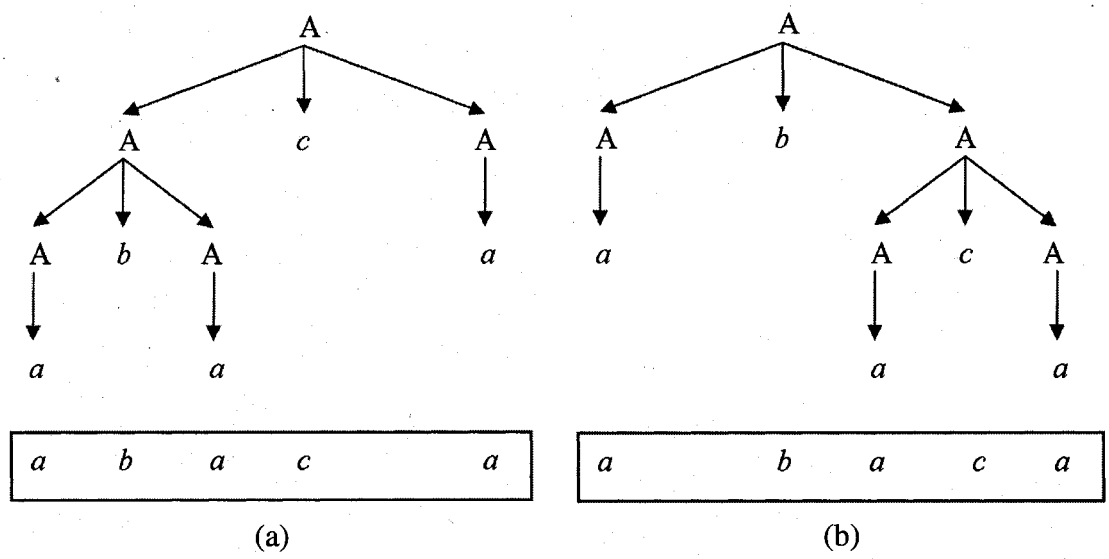


Figure 3 Two derivation tree of the sequence "$a\ b\ a\ c\ a$", given the context-free grammar A $\rightarrow$ A b A | A c A | a. Here, A is identified with the *Start* symbol, even if it also appears on the right side of production rules.

For each sequence $x \in Lg(G)$, let $\overline{\Delta}_x$ represent the set of all possible derivation trees that the grammar G admits, starting with $Start$ and leading to x. Hereafter, $\Delta_x \subset \overline{\Delta}_x$ is some selected subset of derivation trees over x.

For each production rule $A \rightarrow \lambda$ in R, and derivation tree d_x, let $N(A \rightarrow \lambda, d_x)$ denote the number of times that $A \rightarrow \lambda$ appears in d_x. Then the total number of times that the non-terminal symbol A appears in d_x is given by:

$$N(A, d_x) = \sum_{\lambda} N(A \rightarrow \lambda, d_x) \tag{2.1}$$

where the sum is extended over all sentential forms λ for which $A \rightarrow \lambda$ appears in R. In the example tree d_x shown in Fig. 3 (a), one has $N(A, d_x) = 5$ and $N(A \rightarrow a, d_x) = 3$.

At a word level, most MFR systems of interest have a natural and compact description in terms of CFGs. Therefore, a CFG allows to model long term dependencies established between the different words of a MFR sequence. However, given the behavior of MFRs and the imperfections of signals observed on a battlefield, it is not possible to design a robust deterministic CFG to model the behavior of a radar system. To robustly model the signal degradations, noise and uncertainties, an element of stochasticity is introduced into the definition of grammars by assigning probability distributions to the production rules. In *Stochastic Context-Free Grammars* (SCFGs) (Fu, 1982) every production for a non-terminal A has an associated probability value such that a probability distribution exists over the set of productions for A. It incorporates stochastic information that allows for a robust modeling of the signal degradations, noise and uncertainties. SCFGs form an important class of grammars which are widely used to characterize the probabilistic modeling of language in computational linguistic and automatic speech recognition and understanding (Fu, 1982), or in RNA secondary structure prediction (Dowell and Eddy, 2004; Sakakibara *et al.*, 1994).

A SCFG G_s is defined as a pair (G, π) where G is a CFG and $\pi = (\pi_{A_1}, \pi_{A_2}, ..., \pi_{A_r})$ is a vector of probabilities whose each element π_{A_i} represents the distribution of probabilities of a nonterminal A_i producing a combination of symbols λ. So $\theta(A_i \rightarrow \lambda)$ is the probability of A_i producing λ and $\pi_{A_i} = (\theta(A_i \rightarrow \lambda), \theta(A_i \rightarrow \mu), ..., \theta(A_i \rightarrow \sigma))$, where $0 \leq \theta(A_i \rightarrow \lambda) \leq 1$ for λ, and $\sum_\lambda \theta(A_i \rightarrow \lambda) = 1$.

The probability of one derivation d_x of the sequence x of terminal symbols is defined as:

$$P(x, d_x|G_s) = \prod_A \prod_\lambda \theta(A \rightarrow \lambda)^{N(A \rightarrow \lambda, d_x)} \tag{2.2}$$

It corresponds to the product of the probability application functions of all the rules used in the derivation d_x. The *probability of the sequence* x with respect to a specified set of

possible derivations Δ_x is defined as:

$$P(x, \Delta_x|G_s) = \sum_{d_x \in \Delta_x} P(x, d_x|G_s) \tag{2.3}$$

and the *probability of the best derivation of the sequence* x from the set of all derivations Δ_x is defined as:

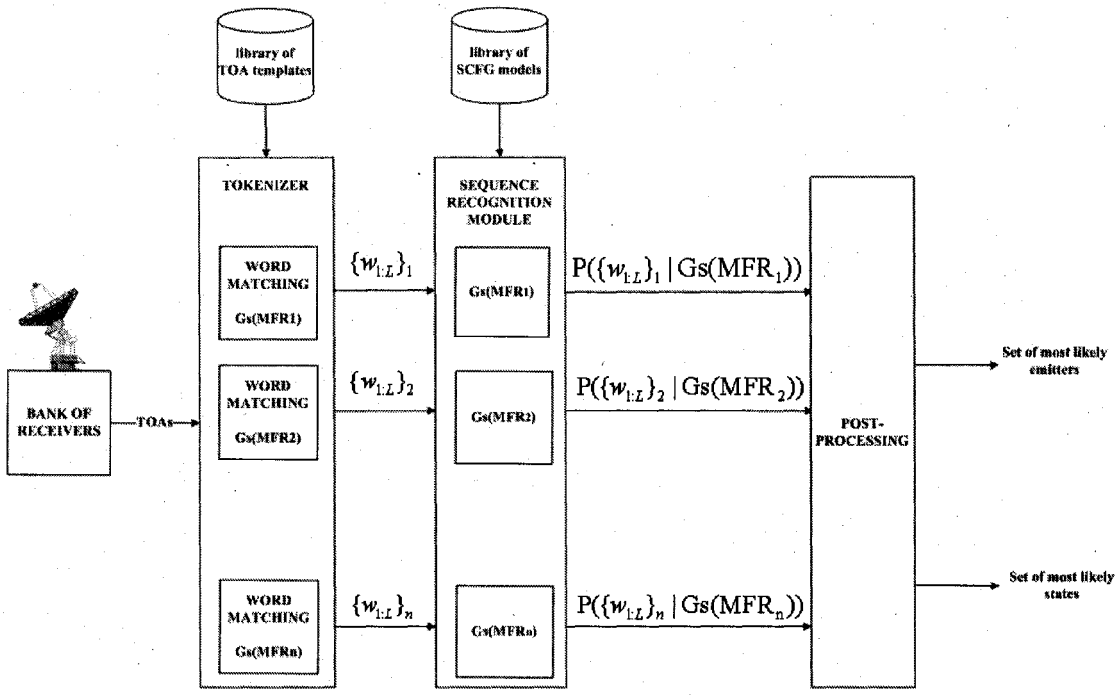
$$\widehat{P}(x|G_s) = \max_{d_x \in \Delta_x} P(x, d_x|G_s) \tag{2.4}$$

Finally, the *best derivation*, $\widehat{d}_x$, is defined as the argument that maximizes Eq. 2.4.

The language $Lg(G_s)$ generated by an SCFG G_s is equal to the language generated by the corresponding CFG G. An important property for any transition probabilities estimation technique is consistency. A SCFG G_s is said to be consistent if $\sum_{x \in Lg(G_s)} P(x|G_s) = 1$ (Sanchez and Benedi, 1997; Fu, 1982).

Fig. 4 shows the block diagram of a radar ES system for recognition of MFRs associated with intercepted pulse trains, and for estimation of the states associated with these MFRs. In this system, a SCFG G_s is used to model each MFR system at a word level only, and therefore would perform the task of sequence recognition and state estimation. In order to perform word recognition, the TOA measured on each incoming pulse sequence is fed to a tokenizer, which performs template matching using, for example, a cross-correlation technique (Dilkes, 2005b; Elton, 2001). Template matching is performed between a window of incoming pulses and the set of words for each MFR. The result is a sequence of words $\{w_{1:L}\}$ for each model of MFR, corresponding to the most likely sequences.

In order to detect the words of a given radar signal, Elton (2001) proposed a cross-correlation (CC) technique. Based on prior information stored in a library, the cross-correlation compares the TOA of pulses in the radar signal with the TOA templates of



(a) ES system.

Figure 4 Block diagram of an ES system that exploits SCFG to model MFR dynamics.

the library. Assuming that the library is composed of the templates that correspond to the words of a radar, the de-interleaving is performed using:

$$R_{sx}(\tau) = \int_{-\infty}^{+\infty} s(t)x(t+\tau)dt \tag{2.5}$$

where $s(t)$ represents the TOA of received pulses and $x(t)$ is TOA template. An example of the signal of pulses produced by this operation for an MFR word is given in Fig. 5. When a sequence of pulses is presented to the tokenizer, the probability of appearance of each MFR word is displayed with respect to the TOA of the pulses. In other words, a peak indicates that a replication of the word begins at the pulse corresponding to this TOA. The input sequence, for this example, corresponds to MFR words that each have a fixed PRI

and a fixed number of pulses. Dilkes (2004b) extends the CC technique for noisy radar data.

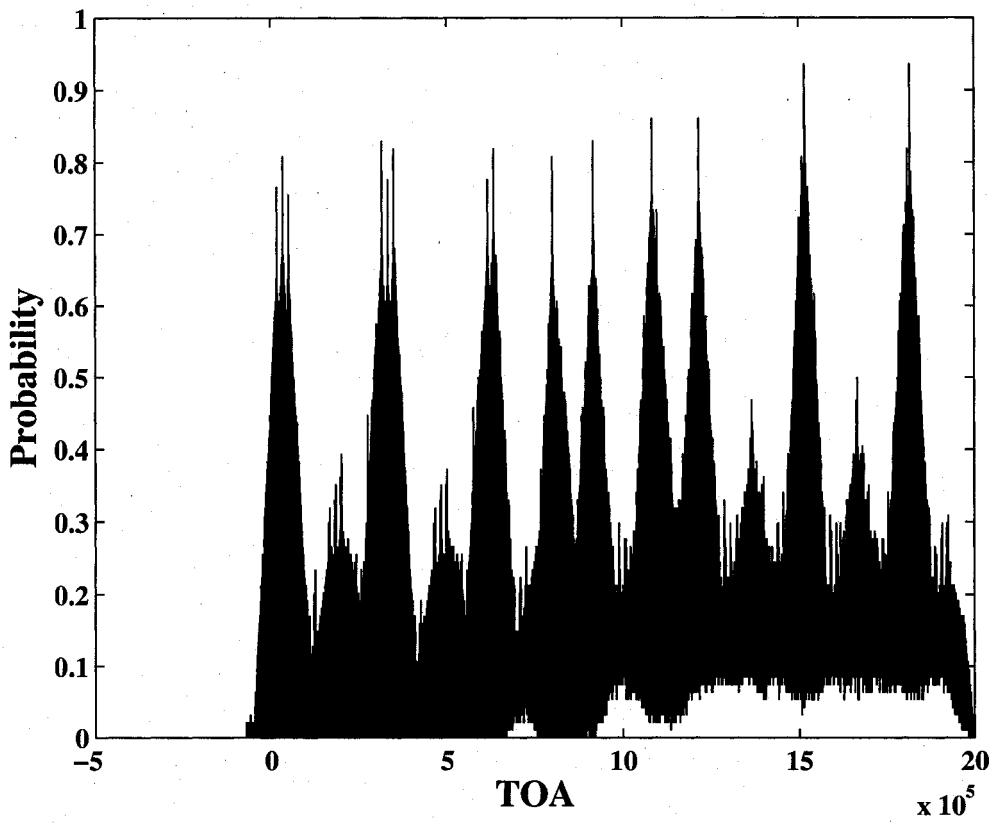


Figure 5 An example of the signal produced for a sequence of pulses that corresponds to a MFR word with fixed PRI, via cross-correlation technique. It represents the probability that the word starts at a TOA, versus the TOA of the pulses.

Once a pattern of pulses is successfully associated with an MFR word, the word replaces the sequence $\{w_{1:L}\}$ corresponding to the MFR. The sequence is then fed to a sequence recognition module. This module computes the probability $P(\{w_{1:L}\}|G_s(MFR))$ that the SCFG G_s associated with each MFR has generated the incoming sequence of MFR words. The sequence recognition module has access to predefined word-level SCFG models, each one corresponding the dynamic behavior of a MFR of interest. If the probability

of a SCFG remains above some pre-defined decision threshold for a sequence of words, one can conclude that it corresponds to the radar type associated with the grammar. In addition, the sequence of words can provide an estimate of that radar's state, and therefore its instantaneous level of threat.

2.4 Learning production probabilities of SCFGs

Although SCFGs provide a natural framework for the description of MFR system, the computational and memory requirements of their signal processing algorithms are generally high. One area of concern is the learning of SCFG rules and/or probability distributions associated with the rules, given some prior knowledge and a set of training sequences. The most popular techniques require very high computational time and memory requirements, that make them unsuitable for radar ES applications. Faster techniques have therefore to be investigated.

SCFGs learning techniques could be integrated into a suite of software tools to assist an ES analyst in the construction of grammatical MFR models for a given theater of operation. The choice of a specific technique depends on the level of prior information to construct the SCFGs. If the analyst knows the basic CFG structure for a MFR of interest, he can learn the production rule probabilities based on a set of training sequences collected in the field. Otherwise, he must also learn the grammatical rules for the CFG (although outside the scope of this report, it is worth noting that grammatical inference techniques have been proposed for learning the rules and probabilities of a SCFG (Sakakibara, 1990) (Nakamura and Matsumoto, 2002)). Finally, if a SCFG has previously been designed, an analyst can incrementally learn a new training sequence that becomes available. If new rules are needed for a new sequence, he can simply add them to the grammar. This has no impact on the previous technique since this rule would not have been used.

This thesis is focused on efficient techniques for learning production rule probabilities of a SCFG. Learning production rule probabilities from training sequences is particularly suit-

able for complex environments, where explicit modeling is difficult. Indeed, the resulting systems can learn and generalize from examples the rules required for MFR recognition. However, their performance depends heavily on the availability of representative training data, and the acquisition of a such a training set is expensive and time consuming in practical applications. Data presented to the ES system in Fig. 4, during either the training or operational phases, may therefore be incomplete in one or more ways.

In ES applications, training data are frequently made available at different points in time. It is therefore highly desirable to update the production rule probabilities of SCFG in an incremental fashion to accommodate the new training data, without compromising the performance. Furthermore, it is not practical in the current setting to accumulate and store all training data in memory, and to retrain a SCFG using all cumulative data. An incremental learning algorithm is the one that meets the following criteria:

a. it should be able to learn additional information from new training sequences;

b. it should not require access to the original training sequences, used to learn the existing SCFG;

c. it should preserve previously-acquired knowledge, i.e., it should not suffer from catastrophic forgetting.

In this thesis, the following approach is considered for designing and maintaining a SCFG to model the dynamics of a MFR:

a. **Initial SCFG design**: define the set of rules of the SCFG to describe the MFR's operation at a word level, and initialize production rule probabilities, either randomly, or based on prior domain knowledge;

b. **Off-line learning**: learn the production rule probabilities parameters of the SCFG based on a representative set of training sequences gathered from the theater of operation;

c. **Incremental learning:** as new training sequences are progressively gathered for the field, perform incremental learning to update and refine existing production rule probabilities of the SCFG based on intercepted sequences from the field. This phase could also involve automatically proposing suitable incremental modifications to the grammatical production rules.

CHAPTER 3

A SURVEY OF TECHNIQUES FOR FAST LEARNING OF PRODUCTION RULE PROBABILITIES

Several different techniques exist for learning the probability distributions associated with the production rules of a SCFG. Estimation of probabilities are typically performed using maximum likelihood (ML) optimization. The most popular ML techniques are the Inside-Outside (IO) algorithm (Baker, 1979; Lari and Young, 1990), that maximizes the likelihood of a dataset and the Viterbi Score (VS) algorithm (Ney, 1992), that maximizes the likelihood of the best derivation trees of a dataset. However, these techniques are far too complex to be of practical use in radar ES applications, which requires timely protection against threats. Several techniques for reducing the time complexities of IO and VS can be found in the literature (see Section 3.2).

In this thesis, a specific type of approach is considered, which is characterized by the use of chart parsers during pre-processing, to accelerate the iterative re-estimation of SCFG probabilities. More precisely, the techniques named Tree Scanning (TS), graphical EM (gEM), and HOLA will be studied. TS and gEM are EM techniques for ML optimization. TS corresponds to the extreme case that lists all the possible derivation trees of the sequences in the training dataset, and can lead to very fast execution for low-ambiguity grammars. However, memory requirements become an issue for high-ambiguity grammars. gEM requires more constant time complexity per iteration, and more moderate use of memory. Both TS and gEM produce the same results as IO. The original versions of these algorithms accelerate IO, and VS derivations of TS and gEM are introduced in this chapter. Finally, HOLA is a gradient descent technique for entropic optimization. These algorithms have been compared and discussed in (Latombe *et al.*, 2006c), (Latombe *et al.*, 2006a), and (Latombe *et al.*, 2006b).

This chapter first presents the classical ML approaches to approximating the probability distributions, and then describes the well-known IO and VS techniques. Then two fast alternatives to IO, namely the Tree Scanning, and the graphical EM techniques, along with their VS derivations, are reviewed. Finally, a gradient descent based technique called HOLA, for optimizing relative entropy is presented. The advantages and drawbacks of these techniques are discussed from an ES perspective.

3.1 Classical EM techniques based on ML approximation

In order to approximate a stochastic distribution defined over the training set, the problem of learning production rule probabilities of a SCFG from a set of sequences can be formulated as an optimization problem. Most popular techniques for optimizing or learning production rule probabilities are based on the EM algorithm, which guarantees that a local maximum is achieved. The objective function depends on the training set and is defined in terms of the probabilities of the rules. It uses growth transformations framework (Sanchez and Benedi, 1997), a special class of function (whose Eq. 3.2 belongs to), to re-estimate SCFG probabilities

Given a SCFG G_s, and any finite collection Ω of training sequences drawn from its language $Lg(G_s)$, with repetitions allowed, the maximum likelihood approach for learning the probabilities of the grammar consists of maximizing an objective function of the form (Nevado *et al.*, 2000):

$$P(\Omega, \Delta_\Omega | G_s) = \prod_{x \in \Omega} P(x, \Delta_x | G_s) \tag{3.1}$$

where $P(x, \Delta_x|G_s)$ is defined in Eq. 2.3. It can be noted that Eq. 3.1 coincides with the *likelihood* of the training sequences (Sanchez and Benedi, 1997) when Δ_x is identified with the set $\overline{\Delta}_x$ consisting of every possible derivation permitted by G_s and leading to $x \in \Omega$, and then $P(x|G_s) = P(x, \overline{\Delta}_x|G_s)$. It also coincides with the likelihood of the *best*

derivation of the sequence when Δ_x contains only the single most probable derivation $\hat{d}_x$ of $x \in \Omega$ (see Eq. 2.4). Both interpretations will be used in the subsequent analysis.

Optimization of Eq. 3.1 is normally implemented using an iterative Expectation-Maximization technique. At each iteration, the following function can be applied to re-estimate the production rule probabilities to approach a local maximum of Eq. 3.1 (Nevado *et al.*, 2000; Sanchez and Benedi, 1997):

$$\theta'(A \rightarrow \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s)}{\sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x | G_s)} \tag{3.2}$$

In Eq. 3.2, if Δ_x represents all the possible derivations $\overline{\Delta}_x$ for each sequence x in the training set, it corresponds to the IO algorithm, while if Δ_x represents only the best derivation $\hat{d}_x$ for each sequence x in the training set, it corresponds to the VS algorithm. It can also be noted that between the IO and VS algorithms, if Δ_x represents the k most probable derivations for each sequence in the training set, then Eq. 3.2 corresponds to the k-best derivation algorithm, or kVS (Nevado *et al.*, 2000; Sanchez and Benedi, 1997).

The rest of this section describes the well-known classical EM algorithms, called IO and VS, in more detail. Each one runs in an iterative manner, by using Eq. 3.2 to modify the probabilities of rules until a local optimum is achieved.

3.1.1 The Inside-Outside (IO) algorithm

The most popular algorithm optimizing the likelihood of the training dataset to re-estimate the probabilities of a SCFG is IO (Baker, 1979; Lari and Young, 1990, 1991), which is based on EM. This algorithm requires the grammar to be in Chomsky Normal Form (Lari and Young, 1990). A CFG is under CNF if its production rules are of the form $A \rightarrow BC$ (which will be called hereafter a transition rule) or $A \rightarrow a$ (which will be called hereafter an emission rule), where $\{A, B, C\}$ are non-terminals and a is a terminal. It is well known

that any CFG can be expressed in Chomsky Normal Form (Hopcroft *et al.*, 2001). As an example, the derivation trees shown in Fig. 3 do not represent a CNF grammar, while those shown in Fig. 8 do.

To re-estimate the probabilities, the IO algorithm computes during each iteration an inside and an outside probability in two passes. To be consistent with the classical notations associated with the CYK parser that will be used for the TS and gEM, all indexes will go from 0 to L. Since the production rules found by the parser will be of the form $A(i,j) \rightarrow B(i,k)C(k,j)$, where $A(i,j)$ indicates that the non-terminal A is at the origin of the subsequence $\{w_{i+1}...w_j\}$ of the parsed sequence $\{w_1...w_L\}$, the inside and outside probabilities of IO follow the same principle in their indexes. An iteration of the IO algorithm may be applied using the following steps:

a. Compute the inside probabilities: Given a training sequence $x = \{w_1, ..., w_L\} \in \Omega$, the inside algorithm computes a probability $\alpha(i-1, j|A) = P(A \Rightarrow w_i, ..., w_j)$ of a sub-tree starting at the non-terminal A and ending at $\{w_i, ..., w_j\}$ as shown in Fig. 6. It can be noted that $\alpha(0, L|Start)$ is the probability of the sequence to be generated by the grammar corresponding to all the possible derivation trees rooted at the initial non-terminal symbol, denoted by $Start$. The algorithm proceeds iteratively using the following recursive relations:

$$\begin{aligned} \alpha(i-1, i|A) &= \theta(A \rightarrow w_i) \\ \alpha(i,j|A) &= \sum_{B \in N} \sum_{C \in N} \sum_{k=i+1}^{j-1} \alpha(i,k|B)\alpha(k,j|C)\theta(A \rightarrow BC), \\ & \qquad \text{for } i < j-1 \end{aligned} \tag{3.3}$$

where N is the set of non-terminals of the grammar, A, B, and C are non-terminals, and w_i is the i^{th} word;

b. Compute the outside probabilities: Given a training sequence $\{w_1, ..., w_L\}$, the outside algorithm computes a probability $\beta(i, j|A) = P(Start \Rightarrow w_1...w_i A w_{j+1}...w_L)$ for all derivation trees containing the non-terminal A that generates the subsequence $\{w_1...w_i\}$ and $\{w_{j+1}...w_L\}$ outside of A (see Fig. 7). The computation of outside probabilities proceeds iteratively using the recursive relations:

$$\beta(0, L|Start) = 1$$

$$\beta(i, j|A) = \sum_{B \in N} \sum_{C \in N} \sum_{k=0}^{i-1} \alpha(k, i|C)\beta(k, j|B)\theta(B \rightarrow CA) + \sum_{B \in N} \sum_{C \in N} \sum_{k=j+1}^{L} \alpha(j, k|C)\beta(i, k|B)\theta(B \rightarrow AC), \quad \text{for } i < j \qquad (3.4)$$

where L is defined as the size of a sequence.

c. Re-estimate the probabilities: IO uses the inside and outside probabilities to re-estimate the probabilities according to Eq. 3.2:

$$\theta'(A \rightarrow BC) = \frac{\sum_{x \in \Omega} \frac{\sum_{0 \le i < k < j \le L} \alpha(i,k|B)\alpha(k,j|C)\beta(i,j|A)\theta(A \rightarrow BC)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}}$$

$$\theta'(A \rightarrow a) = \frac{\sum_{x \in \Omega} \frac{\sum_{i|w_i = a} \beta(i-1,i|A)\theta(A \rightarrow a)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}} \qquad (3.5)$$

Note that α, β, and L depend on the training sequence x.

For reference, the routines for computing inside and outside probabilities, and for reestimating the probabilities are given in Algorithms 1 through 3, in which M_{nt} is the number of non-terminals of the grammar.

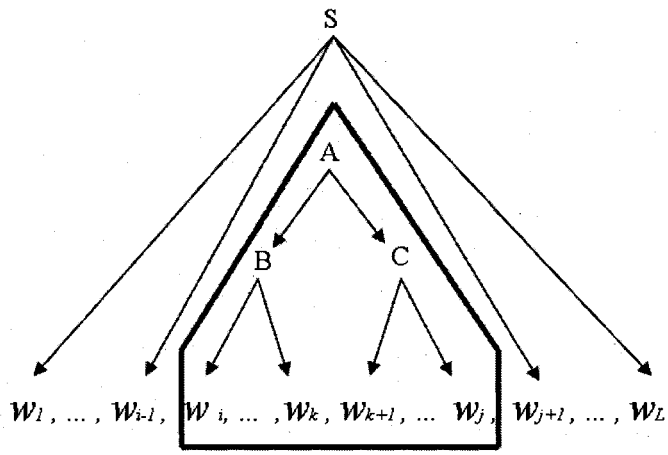


Figure 6 Branches of a SCFG that are relevant for computation of the inside probabilities of the IO algorithm.

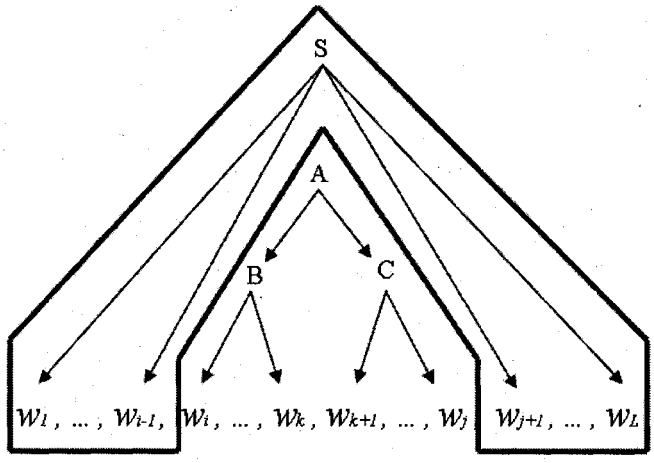


Figure 7 Branches of a SCFG that are relevant for computation of the outside probabilities of the IO algorithm.

Eq. 3.5 follows from Eq. 3.2 due to the following relations:

$$\sum_{d_x \in \bar{\Delta}_x} N(A \rightarrow BC, d_x) P(x, d_x | G_s) = \sum_{0 \leq i < k < j \leq L} \alpha(i,k|B)\alpha(k,j|C)\beta(i,j|A)\theta(A \rightarrow BC)$$

Algorithm 1: `Inside(x)`

%%Initialization%%
for *i=1 to L* **do**
  for $A \in N$ **do**
    $\alpha(i-1, i|A) = \theta(A \rightarrow w_i)$;
%%Iteration%%
for *j=2 to L* **do**
  for *i=j-2 to 0* **do**
    for $A \in N$ **do**
      $\alpha(i,j|A) = \sum_{B \in N} \sum_{C \in N} \sum_{k=i+1}^{j-1} \alpha(i,k|B)\alpha(k,j|C)\theta(A \rightarrow BC)$;

Algorithm 2: `Outside(x)`

%%Initialization%%
$\beta(0, L|Start) = 1$;
for $A \in N \backslash Start$ **do**
  $\beta(0, L|A) = 0$;
%%Iteration%%
for *i=0 to L-1* **do**
  for *j=L to i+1* **do**
    for $A \in N$ **do**
      $\beta(i,j|A) = \sum_{B \in N} \sum_{C \in N} \sum_{k=0}^{i-1} \alpha(k,i|C)\beta(k,j|B)\theta(B \rightarrow CA) +$
      $\sum_{B \in N} \sum_{C \in N} \sum_{k=j+1}^{L} \alpha(j,k|C)\beta(i,k|B)\theta(B \rightarrow AC)$;

$$\begin{aligned} \sum_{d_x \in \bar{\Delta}_x} N(A \rightarrow a, d_x) P(x, d_x|G_s) &= \sum_{i|w_i=a} \beta(i-1, i|A)\theta(A \rightarrow a) \\ \sum_{d_x \in \bar{\Delta}_x} N(A, d_x) P(x, d_x|G_s) &= \sum_{0 \le i < j \le L} \alpha(i,j|A)\beta(i,j|A) \end{aligned} \quad (3.6)$$

Consider the example shown in Fig. 8. This sequence corresponds to a phrase from an MFR of type Mercury, and will be used to train Mercury using IO. The Mercury Detection-Level Grammar given in Annex 3 is considered to have been initialized in a uniform way, which means that given a non-terminal A, every $\theta(A \rightarrow \lambda)$ is equal if $A \rightarrow \lambda$ appears

Algorithm 3: `Inside-Outside()`

while $loglikelihood(m) - loglikelihood(m-1) > \epsilon$ **do**
  for $x \in \Omega$ **do**
    $\alpha_x = \texttt{Inside}(x)$;
    $\beta_x = \texttt{Outside}(x)$;
  $loglikelihood(m) = \sum_{x \in \Omega} \alpha_x(0, L(x)|Start)$;
  for $A \in N$ **do**
    for $B \in N$ **do**
      for $C \in N$ **do**

$$\theta'(A \rightarrow BC) = \frac{\sum_{x \in \Omega} \frac{\sum_{0 \leq i < k < j \leq L} \alpha(i,k|B)\alpha(k,j|C)\beta(i,j|A)\theta(A \rightarrow BC)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \leq i < j \leq L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}};$$

    for $a \in V$ **do**

$$\theta'(A \rightarrow a) = \frac{\sum_{x \in \Omega} \frac{\sum_{i|w_i = a} \beta(i-1,i|A)\theta(A \rightarrow a)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \leq i < j \leq L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}};$$

  $m = m + 1$;

in the grammar definition. Then, the production probabilities appearing in Fig. 8 are the following:

$$\begin{aligned}
&\theta(Start \rightarrow \texttt{Acq}\ \epsilon) = 0.1 \\
&\theta(Start \rightarrow \texttt{Na}\ \epsilon) = 0.1 \\
&\theta(Start \rightarrow \texttt{Tm}\ \epsilon) = 0.1 \\
&\theta(\texttt{Acq} \rightarrow Q6\ Q6) = 0.1667 \\
&\theta(\texttt{Na} \rightarrow Q6\ Q6) = 0.5 \\
&\theta(\texttt{Tm} \rightarrow Q6\ Q6) = 0.1408 \\
&\theta(Q6 \rightarrow W6\ W6) = 1 \\
&\theta(W6 \rightarrow 6) = 1 \\
&\theta(\epsilon \rightarrow 10) = 1
\end{aligned} \tag{3.7}$$

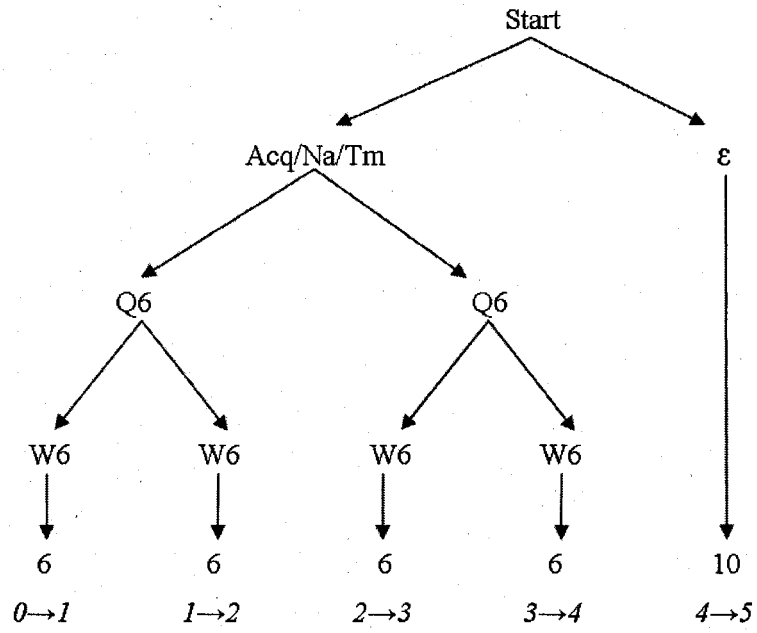


Figure 8 Example of a derivation tree from a short sequence emitted by an MFR.

Suppose that the quantities $\alpha(0,4|\texttt{Acq}) = 0.1667$, $\alpha(0,4|\texttt{Na}) = 0.15$, $\alpha(0,4|\texttt{Tm}) = 0.1408$ and $\alpha(4,5|\epsilon) = 1$ have already been computed. Then, according to Eq. 3.3, $\alpha(0,5|Start)$ will be computed as follows:

$$\begin{aligned}\alpha(0,5|Start) &= \alpha(0,4|\texttt{Acq})\alpha(4,5|\epsilon)\theta(Start \rightarrow \texttt{Acq}\ \epsilon) + \alpha(0,4|\texttt{Na})\alpha(4,5|\epsilon) \\ &\quad \theta(Start \rightarrow \texttt{Na}\ \epsilon) + \alpha(0,4|\texttt{Tm})\alpha(4,5|\epsilon)\theta(Start \rightarrow \texttt{Tm}\ \epsilon) \\ &= 0.1667 \cdot 1 \cdot 0.1 + 0.5 \cdot 1 \cdot 0.1 + 0.1408 \cdot 1 \cdot 0.1 = 0.08075 \quad (3.8)\end{aligned}$$

And according to Eq. 3.4, $\beta(0,4|\texttt{Acq})$ will be computed as follows:

$$\begin{aligned}\beta(0,4|\texttt{Acq}) &= \alpha(4,5|\epsilon)\beta(0,5|Start)\theta(Start \rightarrow \texttt{Acq}\ \epsilon) \\ &= 1 \cdot 1 \cdot 0.1 = 0.1 \quad (3.9)\end{aligned}$$

The reestimation formula of Eq. 3.5 for $\theta'(Start \rightarrow \texttt{Acq}\ \epsilon)$ gives:

$$\begin{aligned}\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) &= \frac{\frac{1}{\alpha(0,5|Start)}\ \alpha(0,4|\texttt{Acq})\alpha(4,5|\epsilon)\beta(0,5|Start)\theta(Start \rightarrow \texttt{Acq}\ \epsilon)}{\frac{1}{\alpha(0,5|Start)}\ \alpha(0,5|Start)\beta(0,5|Start)} \\ &= \frac{\frac{1}{0.08075}\ 0.1667 \cdot 1 \cdot 1 \cdot 0.1}{\frac{1}{0.08075}\ 0.08075 \cdot 1} = 0.20644 \qquad (3.10)\end{aligned}$$

When IO estimates $\alpha(i,j|A)$ and $\beta(i,j|A)$, it passes through every possible combination of non-terminals $A \rightarrow BC$ as though each non-terminal could produce any pair of non-terminals, even if, according to the grammar, BC cannot be produced by A. Moreover, it considers that any non-terminal can have non-null inside and outside probabilities, whatever the subsequence. This will result in a time complexity per iteration of $O(M_{nt}^3 \cdot L^3)$, which increases exponentially with the number of non-terminals and the size of the sequence. On the other hand, it has the advantage of having a low memory complexity of $O(L^2 \cdot M_{nt})$.

It should be noted that for some applications, this algorithm can also be used for grammatical inference. For example, one can use IO with grammars that have different numbers of non-terminals, in which no probability is set to 0, and then select the best number of non-terminals (Lari and Young, 1990, 1991).

3.1.2 The Viterbi Score (VS) algorithm

While IO seeks to maximize the likelihood of a training set, the VS (Ney, 1992) algorithm, seeks to maximize the likelihood of the best derivations of a training set. Once a grammar is in CNF format, an iteration of the VS algorithm may be applied using the following steps:

a. Find the most likely derivation tree for the corresponding sequence: Let $\hat{\alpha}(i,j|A)$ represent the largest inside probability of any subtree rooted at a non-terminal

A, and generating the subsequence $w_{i+1}, ..., w_j$. Let $\psi(i, j|A)$ contain the rule $A(i, j) \rightarrow B(i, k)C(k, j)$ representing the highest vertex of the most probable subtree, including the non-terminals B and C and the word index k. For compact representation, let $\psi(i, j|A)$ refer only to the vector $[B, C, k]$, since it is enough to retrace the rule $A(i, j) \rightarrow B(i, k)C(k, j)$. These can be computed recursively using the following equations:

$$\begin{aligned}
&\hat{\alpha}(i, i+1|A) = \theta(A \rightarrow w_{i+1}) \\
&\hat{\alpha}(i, j|A) = \max_{B,C} \max_{i<k<j} \{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}, \quad \text{for } i < j-1 \\
&\psi(i, j|A) = [B, C, k] = \text{argmax}_{B,C,k}\{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } i < j-1 \quad (3.11)
\end{aligned}$$

The derivation tree $\hat{d}_x$ can now be retraced by starting at $\psi(0, L|Start)$;

b. Count the number of times each rule appears in the derivation tree found from $\psi(0, L|Start)$;

c. Re-estimate the probabilities:

$$\text{If } \sum_{x\in\Omega} N(A, \hat{d}_x) \neq 0, \qquad \theta'(A \rightarrow \lambda) = \frac{\sum_{x\in\Omega} N(A \rightarrow \lambda, \hat{d}_x)}{\sum_{x\in\Omega} N(A, \hat{d}_x)} \qquad (3.12)$$

In the case of VS, Eq. 3.2 reduces to Eq. 3.12 since Δ_x contains only the best derivation tree $\hat{d}_x$.

For reference, the routines for finding the best derivation can be found in Algorithms 4 through 6. Algorithm 7 allows counting the frequency of the production rules and reestimating the probabilities.

Algorithm 4: `Maximum-Probability(x)`

%%Initialization%%
for *i=1 to L* **do**
  for $A \in N$ **do**
    $\hat{\alpha}(i-1, i|A) = \theta(A \rightarrow w_i)$;
%%Iteration%%
for *j=2 to L* **do**
  for *i=j-2 to 0* **do**
    for $A \in N$ **do**
      $\hat{\alpha}(i, j|A) = \max_{B,C \in N} \max_{i<k<j} \{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}$;
      $\psi(i, j|A) = \text{argmax}_{B,C,k} \{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}$;

Algorithm 5: `Retrace-path(x)`

$store_1^x = Start \rightarrow \psi(0, L|Start)(1)\psi(0, L|Start)(2)$;
$i = 0$;
$j = L$;
$k = \psi(0, L|Start)(3)$;
$B = \psi(0, L|Start)(1)$;
$C = \psi(0, L|Start)(2)$;
if $k - i > 1$ **then**
  `Add-store`$(\psi(i, k|B))$;
if $j - k > 1$ **then**
  `Add-store`$(\psi(k, j|C))$;

Algorithm 6: `Add-store` $(\psi(i, j|A))$

$store_{end+1}^x = A \rightarrow \psi(i, j|A)(1)\psi(i, j|A)(2)$;
$k = \psi(i, j|A)(3)$;
$B = \psi(i, j|A)(1)$;
$C = \psi(i, j|A)(2)$;
if $k - i > 1$ **then**
  `Add-store`$(\psi(i, k|B))$;
else
  $store_{end+1}^x = B \rightarrow w_k$;
if $j - k > 1$ **then**
  `Add-store`$(\psi(k, j|C))$;
else
  $store_{end+1}^x = C \rightarrow w_j$;

Algorithm 7: `Viterbi-Score()`

initialize $histo_t$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
initialize $histo_e$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
%%Histograms%%
for $x \in \Omega$ **do**
  for *i=1 to* $|store^x|$ **do**
    if $|store_i^x|$=*3* **then**
      $histo_t(store_i^x(1), store_i^x(2), store_i^x(3)) = histo_t(store_i^x(1), store_i^x(2), store_i^x(3)) + 1$;
    else
      $histo_e(store_i^x(1), store_i^x(2)) = histo_e(store_i^x(1), store_i^x(2)) + 1$;
%%Reestimation%%
for $A \in N$ **do**
  for $B \in N$ **do**
    for $C \in N$ **do**
      $\theta(A \rightarrow BC) = \frac{histo_t(A,B,C)}{\sum_{iD \in N} \sum_{E \in N} histo_t(A,D,E)}$;
  for $a \in V$ **do**
    $\theta(A \rightarrow a) = \frac{histo_e(A,a)}{\sum_{b \in V} histo_e(A,b)}$;

Consider the example of Fig. 8. According to the probabilities, the maximum likelihood derivation tree $\hat{d}_x$ is shown on Fig. 9, and the corresponding frequency of the rules is given in Eq. 3.13.

$$\begin{aligned} N(\texttt{Na} \rightarrow Q6\ Q6, \hat{d}_x) &= 1 \\ N(\texttt{Acq} \rightarrow Q6\ Q6, \hat{d}_x) &= 0 \\ N(\texttt{Tm} \rightarrow Q6\ Q6, \hat{d}_x) &= 0 \end{aligned} \tag{3.13}$$

Since `Na` can lead to other couples of non-terminals, $\theta(\texttt{Na} \rightarrow Q6\ Q6)$ has then to be normalized according to Eq. 3.12.

$$\theta'(\mathtt{Na} \rightarrow Q6\ Q6) = \frac{N(\mathtt{Na} \rightarrow Q6\ Q6, \hat{d}_x)}{\sum_\lambda N(\mathtt{Na} \rightarrow \lambda, \hat{d}_x)} \tag{3.14}$$

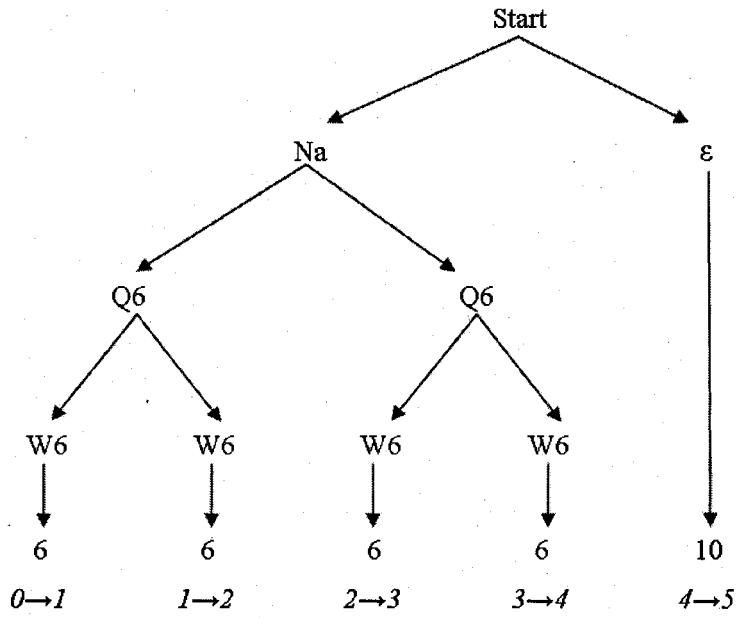


Figure 9 Best derivation tree for example in Fig. 8.

Both IO and VS have a worst-case time complexity per iteration that grows with $O(M_{nt}^3 L^3)$, and a memory complexity that grows with $O(L^2 M_{nt})$. However, it can be seen that, contrary to IO, VS requires only one pass to re-estimate the probabilities, and this gives a lower time complexity per iteration in practice. In addition VS is known to converge with fewer iterations than IO, even though the SCFGs are not, in general, as well learned (Sanchez and Benedi, 1999a).

Other alternatives to IO and VS include (1) the k-best derivation algorithm and (2) the VS algorithm with prior information. As they were not implemented for simulations, they are simply described in Annex 2.

3.2 Fast learning techniques based on ML approximation

Several alternatives have been proposed to accelerate the SCFG learning in different applications. Sakakibara (1990; 1993; 1994) uses Tree-Grammars, a technique to present the data such that it avoids passing through all the possibilities. Probability reestimation is then achieved using a generalization of HMM forward-backward algorithm and leads to a time complexity of $O(L^3 + M_{nt}^3 L)$. Kupiec (1992) uses an Hidden Markov Model (HMM) based representation of the grammar with trellis diagrams in order to compute the IO probabilities. For a same grammar, this algorithm has the same complexity as IO, but does not require the grammar to be in Chomsky Normal Form (CNF), which reduces the number of non-terminals M_{nt} and therefore results in a lower time complexity. Lucke (1994) proposes a BLI – the author does not define this acronym – algorithm in which the probabilities are approximated in a manner that is applicable to IO. It uses a stochastic parser, and perceives a derivation tree as a Bayesian network. Probabilities are re-estimated using two vectors called the evidential and causal support of a node. The approximations allow him to reduce IO time complexity from $O(M_{nt}^3 L^3)$ to $O(M_{nt}^2 L^3)$. Ito *et al.* (2001) reduces the time complexity from $O(M_{nt}^3)$ to $O(M_{nt}^2)$ by using restricted grammars, in which rules are of the form $A \rightarrow AB$ or $A \rightarrow BA$. The author explains that this kind of grammar can model many languages, including English. Finally, Chen and Chaudhari (2003) propose to use a prediction function before applying IO in order to detect some redundant operations. Modifying IO to avoid these operations allows reducing the time complexity per iteration by some unspecified amount.

In this thesis, a popular type of approach is considered to reduce the *time complexity per iteration* of IO. It involves pre-computing data structures such as support graphs and histograms, using tools like the Earley (Earley, 1970) or Cocke-Younger-Kasami (CYK) (Nijholt, 1991; Hopcroft *et al.*, 2001) parsers during the pre-processing phase. Pre-computation of data structures may then accelerate the iterative probability re-estimation process, since the blind combination of rules, where any non-terminal symbol could pro-

duce any combination of non-terminals, is avoided. All these techniques may in practice give lower time complexity, at the expense of an increased memory complexity. They are preferred in this work because MFR grammars are very simple (much more than natural language grammars for example). Thus, approaches that accelerate IO using the grammars structure rather than more general algorithmic improvement seems more appropriate. Fujisaki *et al.* (1989) were the first author to adapt this approach. They proposed using a CYK parser to find the derivations or the most probable derivations of the sentences, and then directly apply Eq. 3.2, corresponding either the Inside-Outside or the Viterbi algorithm. Based on this work, an algorithm called from now on Tree Scanning (TS) (Latombe *et al.*, 2006b), where all the possible derivation trees corresponding to a training set are computed in order to apply the basic reestimation equations, has been introduced. Two versions of the algorithms, one for IO,named TS(IO), and the other for VS, named TS(VS), have been proposed. If this algorithm is faster than IO in most of the applications, it has a time complexity of $O(M_{nt}^{L} \cdot L^3)$, and a memory complexity of $O(M_{nt}^{L} \cdot L)$ for a grammar of unbounded ambiguity. TS usually corresponds to the case where the most memory is sacrificed to accelerate time complexity.

Stolcke (1995) proposed an algorithm that computes the inside and outside probabilities of IO during the steps of an Earley parsing. However, this algorithm requires two passes – one for the inside probability and one for the outside probability. It has the same time complexity per iteration of $O(M_{nt}^{3} \cdot L^3)$ in the worst case, that is reduced to $O(M_{nt}^{3} \cdot L^2)$ for grammars of bounded ambiguity, and a memory complexity of $O(M_{nt}^{3} \cdot L^2)$.

Ra and Stockman (1999) introduced an extension of this last technique that computes both inside and probabilities in only one pass, using a special term that stores the weighted count of the rules appearing during the parsing, but increasing space complexity drastically to $O(M_{nt}^{6} \cdot L^2)$. Time complexity then becomes $O(\|r\|^2 L^3)$, where $\|r\|$ is the number of rules of the grammars. However, in the worst case, it is $O(M_{nt}^{6} L^3)$, which is more than

IO's. In this work, Stolcke and Ra's approaches are still too complex to be of practical use and were therefore not implemented.

More recently, Sato and Kameya (2001) initially used a chart parser to produce a special representation of the training data called support graphs, where only the combination of rules leading to the analyzed sequence are represented. Then, they run a new IO-like algorithm based on these support graphs called graphical EM (gEM). In literature, only an IO version of the algorithm is developed, a technique identified here in as gEM(IO). Therefore, a new version of gEM, named gEM(VS), that allows for applying VS to gEM, is proposed here. Their experiments show a five-fold reduction in time complexity per iteration on the ATR corpus (Uratani *et al.*, 1994), composed of 10995 short and conversational Japanese sentences. It will be shown that its time complexity per iteration and its memory complexity grow with $O(M_{nt}^3 \cdot L^3)$. It is worth noting that the techniques proposed by Fujisaki, Stolcke, Ra and Stockman, and Sato and Kameya compute the same values as IO and provide exactly the same results as IO.

Oates and Heeringa (2002) have introduced a heuristic incremental gradient descent algorithm called HOLA – the authors do not define this acronym – based on summary statistics. It uses a standard chart parser to compute the distributions of rules (the summary statistics) found after parsing the training database and after parsing a set of sequences produced by the grammar. Re-estimation of probabilities is performed using an approximation of the gradient descent (Annex 3.3). Unlike the other techniques, HOLA does not optimize explicitly the likelihood of a sequence (as with IO), but rather the relative entropy between these two distributions. It has the advantage of having very low time complexity per iteration and memory complexity of $O(M_{nt}^3)$.

The rest of this subsection presents a more detailed description of the TS, gEM, and HOLA algorithms.

3.2.1 The Tree Scanning (TS) Technique

The Tree Scanning (TS) algorithm consists in using a chart given by a classic Earley (Earley, 1970) or CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) parser to find all the possible trees producing a sequence and then applying Eq. 3.2. In most cases it represents the limiting case in trading off time complexity for memory complexity. A possible pseudo-code to extract the trees after Earley parsing is given in Algorithms 8 and 9, and a possible pseudo-code to extract trees from a CYK chart is given in Algorithms 11 and 12.

Algorithm 8: `Earley-Extract()`

arrange *store*, the result from Earley parsing containing the rules (see Algorithm 28) by *stacks*, where each *stack* corresponds to the initial word of the subsequence stepped by the rule (see Annex 1 for more details on the Earley parser);
initialize the tools $dep1 = dep2 = [1]$;
`NewTree();`
while *stop=0* **do**
  foreach *stack* **do**
    `TreeCompletion();`
    `NewTree();`
    if *no element of dep1 appears in the stack as the producer of a rule* **then**
      $stop = 1$;

Algorithm 9: `NewTree()`

for *i=1 to* $|d_x|$ **do**
  for *m=1 to* $|dep2|$ **do**
    for *n=1 to* $|stack|$ **do**
      if $stack_n$ *is a transition rule* **then**
        $A \rightarrow BC = stack_n$;
      else
        $A \rightarrow a = stack_n$;
      if *A=deptemp(m)* **then**
        $d_x^{end+1} = d_x^i$;
        $dep1_{end+1} = dep1_i$;
$dept2 = dep1$;
remove all the redundant indexes from $dep2$;

Algorithm 10: `TreeCompletion()`

```
for i=1 to |d_x| do
    for m=1 to |dep1| do
        for n=1 to |stack| do
            if stack_n is a transition rule then
                A → BC = stack_n;
            else
                A → a = stack_n;
            if A=dep(m) then
                if stack_n is a transition rule then
                    add A → BC to d_x;
                    add B and C to dep1;
                else
                    add A → a to d_x;
```

Algorithm 11: `CYK-Extract()`

```
condition = 1;
stop = 0;
start = 1;
I = 0;
CYK-Trees(Start(0,L));
while stop=0 do
    start2=|d_x|+1;
    for l=start to |d_x| do
        A(i,j) → B(i,k)C(k,j) = chart_{d_x^{l,end}(4), d_x^{l,end}(5)}(I(l));
        CYK-Trees(B(i,k),l);
        CYK-Trees(C(k,j),l);
    if |d_x|=condition then
        stop = 1;
    condition=|d_x|;
    start = start2;
```

In both cases, the result is a set of stored rules that correspond to the derivation trees d_x in the algorithms. It is then easy to apply Eq. 3.2 according to the desired method (IO, VS or kVS).

Once the trees corresponding to the sequences of a training data set have been computed, an iteration of the TS algorithm may be applied using the following steps:

Algorithm 12: `CYK-Trees(A(i,j),numTree)`

count=0;
for *i=1 to* $|chart_{i,j}|$ **do**
  if *the rule corresponding to* $chart_{i,j}(i)$ *expands* A **then**
    $count = count + 1$;
    if *count=1* **then**
      $index = i$;
      if $chart_{i,j}(i)$ *is of the form* $B(k,m) \rightarrow C(k,l)D(l,m)$ **then**
        add $B \rightarrow C, D$ to $d_x^{numTree}$;
      else
        add $B \rightarrow Wi$ to $d_x^{numTree}$;
    else
      %%there is a new derivation: addition of a new tree%%
      $d_x^{end+1} = d_x^{numTree}$;
      if $chart_{i,j}(i)$ *is of the form* $B(k,m) \rightarrow C(k,l)D(l,m)$ **then**
        add $B \rightarrow C, D$ to d_x^{end};
      else
        add $B \rightarrow Wi$ to d_x^{end};
      $I(end+1) = i$;
    $resumption$=$|d_x|$;
    $B(k,m) \rightarrow C(k,l)D(l,m) = chart_{i,j}(index)$;
    if $d_x^{numTree}(resumption) = B \rightarrow CD$ **then**
      `CYK-Trees`$(C(k,l), numTree)$;
      `CYK-Trees`$(D(l,m), numTree)$;

a. Count the frequency of the rules and compute the probabilities of the trees:

   Use Eq. 2.2;

b. Re-estimate the probabilities: Use Eq. 3.2.

These two steps are performed using Algorithm 13 for the IO version of TS, named TS(IO)[1], and using Algorithm 14 for the VS version of TS, named TS(VS). The symbols

[1] Note that TS does not compute Inside and Outside probabilities. The notation just refers to the fact that all the derivation trees are considered during the re-estimation of the probabilities.

Algorithm 13: `Tree-Scanning(IO)`

while $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  for *x=1 to* $|\Omega|$ **do**
    %%Histograms%%
    initialize $histo\_t_x$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
    initialize $histo\_e_x$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
    for *i=1 to* $|d_x|$ **do**
      for *j=1 to* $|d_x^i|$ **do**
        if $|d_x^i(j)|$*=3* **then**
          $histo\_t_{x,i}(d_x^i(j)) = histo\_t_{x,i}(d_x^i(j)) + 1$;
        else
          $histo\_e_{x,i}(d_x^i(j)) = histo\_e_{x,i}(d_x^i(j)) + 1$;
    %%Probability computation for each tree%%
    $p_{x,1to|d_x|} = 1$;
    for *i=1 to* $|d_x|$ **do**
      for *j=1 to* $|d_x^i|$ **do**
        $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;
    $prod\_t_x = \sum_i p_{x,i} \cdot histo\_t_{x,i}$;
    $prod\_e_x = \sum_i p_{x,i} \cdot histo\_e_{x,i}$;
    $ptotal_x = \sum_i p_{x,i}$;
  $cond^{(m)} = \sum_{x,i} p_{x,i}$;
  %%Reestimation%%
  $num\_t = \sum_x \frac{prod\_t_x}{ptotal_x}$;
  $num\_e = \sum_x \frac{prod\_e_x}{ptotal_x}$;
  for $i = 1$ *to* M_{nt} **do**
    $denom(i) = \sum_{j=1}^{M_{nt}} \sum_{k=1}^{M_{nt}} num\_t(i,j,k) + \sum_{j=1}^{M_t} num\_e(i,j)$;
  foreach *rule* $A \rightarrow BC$ *or* $A \rightarrow a$ **do**
    $\theta'(A \rightarrow BC) = \frac{num\_t(A,B,C)}{denom(A)}$;
    $\theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}$;
  $m = m + 1$;

used in Alg. 13 can be linked to Eq. 3.2 as follows:

$$
\begin{aligned}
num_t(A,B,C) &= \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow BC, d_x) P(x, d_x | G_s) \\
num_e(A,a) &= \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow a, d_x) P(x, d_x | G_s)
\end{aligned}
\tag{3.15}
$$

Algorithm 14: `Tree-Scanning(VS)`

for *x=1 to* $|\Omega|$ **do**
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;

while $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  for *x=1 to* $|\Omega|$ **do**
    %%Histograms%%
    initialize $histo\_t_x$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
    initialize $histo\_e_x$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
    for *i=1 to* $|d_x|$ **do**
      for $j = argmax_i\{p_{x,i}\}$ **do**
        if $|d_x^i(j)|$*=3* **then**
          $histo\_t_x(d_x^i(j)) = histo\_t_x(d_x^i(j)) + 1$;
        else
          $histo\_e_x(d_x^i(j)) = histo\_e_x(d_x^i(j)) + 1$;
  %%Reestimation%%
  $num\_t = \sum_x histo\_t_x$;
  $num\_e = \sum_x histo\_e_x$;
  for $i = 1$ *to* M_{nt} **do**
    $denom(i) = \sum_{j=1}^{M_{nt}} \sum_{k=1}^{M_{nt}} num\_t(i,j,k) + \sum_{j=1}^{M_t} num\_e(i,j)$;
  foreach *rule* $A \rightarrow BC$ *or* $A \rightarrow a$ **do**
    $\theta'(A \rightarrow BC) = \frac{num\_t(A,B,C)}{denom(A)}$;
    $\theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}$;
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_i = p_i \cdot \theta(d_x^i(j))$;
  $cond^{(m)} = \sum_x \max_i\{p_{x,i}\}$;
  $m = m + 1$;

$$
\begin{aligned}
denom(A) &= \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x | G_s) \\
\theta'(A \rightarrow BC) &= \frac{num\_t(A,B,C)}{denom(A)} \qquad \theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}
\end{aligned} \tag{3.16}
$$

Consider the example of Fig. 8. Three different trees lead to the sequence $x = 6\ 6\ 6\ 6\ 10$:

$$d_x^1 = [Start, \texttt{Acq}, \epsilon][\texttt{Acq}, Q6, Q6][Q6, W6, W6][W6, 6][W6, 6][Q6, W6, W6] \\ [W6, 6][W6, 6][\epsilon, 10]$$

$$d_x^2 = [Start, \texttt{Na}, \epsilon][\texttt{Na}, Q6, Q6][Q6, W6, W6][W6, 6][W6, 6][Q6, W6, W6] \\ [W6, 6][W6, 6][\epsilon, 10]$$

$$d_x^3 = [Start, \texttt{Tm}, \epsilon][\texttt{Tm}, Q6, Q6][Q6, W6, W6][W6, 6][W6, 6][Q6, W6, W6] \\ [W6, 6][W6, 6][\epsilon, 10]$$

where, for example, $[Start, \texttt{Tm}, \epsilon]$ represents the rule $Start \rightarrow \texttt{Tm}\ \epsilon$.

From these trees, it is easy to find $P(x, \Delta_x|G_s) = \alpha(0, 5|Start)$, using Eq. 2.2 and 2.3 with the production probabilities of Eq. 3.7:

$$\begin{aligned} P(x, \Delta_x|G_s) &= P(x, d_x^1|G_s) + P(x, d_x^2|G_s) + P(x, d_x^3|G_s) \\ &= \prod_{A\rightarrow\lambda} \theta(A \rightarrow \lambda)^{N(A\rightarrow\lambda, d_x^1)} + \prod_{A\rightarrow\lambda} \theta(A \rightarrow \lambda)^{N(A\rightarrow\lambda, d_x^1)} \\ &\quad + \prod_{A\rightarrow\lambda} \theta(A \rightarrow \lambda)^{N(A\rightarrow\lambda, d_x^1)} \\ &= 0.1 \cdot 0.1667 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 + 0.1 \cdot 0.5 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \\ &\quad + 0.1 \cdot 0.1408 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \\ &= 0.01667 + 0.05 + 0.01408 = 0.08075 \end{aligned} \tag{3.17}$$

For TS(IO), like for classical IO, $\theta(Start \rightarrow \texttt{Acq}\ \epsilon)$ can be re-estimated:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = \frac{\sum_{x\in\Omega} \frac{1}{P(x,\Delta_x|G_s)} \sum_{d_x\in\Delta_x} N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x) P(x, d_x|G_s)}{\sum_{x\in\Omega} \frac{1}{P(x,\Delta_x|G_s)} \sum_{d_x\in\Delta_x} N(Start, d_x) P(x, d_x|G_s)} \tag{3.18}$$

where

$$\sum_{d_x \in \Delta_x} N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x) P(x, d_x^1 | G_s) = N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x^1) P(x, d_x^1 | G_s) + N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x^2) P(x, d_x^2 | G_s) + N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x^3) P(x, d_x^3 | G_s) \tag{3.19}$$

and

$$\sum_{d_x \in \Delta_x} N(Start, d_x) P(x, d_x | G_s) = N(Start, d_x^1) P(x, d_x^1 | G_s) + N(Start, d_x^2) P(x, d_x^2 | G_s) + N(Start, d_x^3) P(x, d_x^3 | G_s) \tag{3.20}$$

which gives:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = \frac{\frac{1}{0.08075} \cdot (1 \cdot 0.01667 + 0 \cdot 0.05 + 0 \cdot 0.01408)}{\frac{1}{0.08075} \cdot (1 \cdot 0.01667 + 1 \cdot 0.05 + 1 \cdot 0.01408)} = 0.20644 \tag{3.21}$$

For TS(VS), like for classical VS, $\theta(Start \rightarrow \texttt{Acq}\ \epsilon)$ can be re-estimated:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = \frac{N_{\hat{d_x}}(Start \rightarrow \texttt{Acq}\ \epsilon)}{N_{\hat{d_x}}(Start)} = \frac{N_{d_x^2}(Start \rightarrow \texttt{Acq}\ \epsilon)}{N_{d_x^2}(Start)} = \frac{0}{1} = 0 \tag{3.22}$$

This algorithm has the practical advantage of being very fast once the data has been pre-processed, when ambiguity is of low order of magnitude. Moreover the data from the pre-processing is very easy to store, as it does not require any particular order in the organization of rules. Although the memory requirements needed with the pre-processing are very low for low-ambiguity grammars, they become an issue for high-ambiguity grammars. It has the disadvantage, when ambiguity rises, of having a time complexity per iteration and a memory complexity that grow with $O(M_{nt}^{L} \cdot L^3)$ and $O(M_{nt}^{L} \cdot L)$, respectively.

3.2.2 The Graphical EM algorithm

During pre-processing, this algorithm creates a set of ordered support graphs from the chart of an Earley (Earley, 1970) or CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) parser, to represent only the derivations that may possibly lead to a sequence. For the following, we will assume that a CYK parser has initially been used on the data (cf. Annex 1). From the resulting chart T, the algorithm creates support graphs, each one showing the possible productions from a non-terminal that generates the subsequence from w_{i+1} to w_j. Since the creation of the support graphs begins with $Start$, only production rules existing in the derivation trees leading to a given sequence will appear. The support graphs should be ordered such that a "father" is always before a "son", and the support graph generated by $Start$ is always first. Support graphs are extracted using routines `Extract-CYK` and `Visit-CYK` presented in Algorithms 15 and 16. Fig. 10 shows the CYK chart corresponding to the example in Fig. 8. Fig. 11 shows an example of support graphs created from the chart of Fig. 10, along with the corresponding notations.

During the iterative process gEM operates in a similar way to the IO algorithm. The main difference lies in the fact that gEM only passes by the transitions described in support graphs to re-estimate the probabilities. Once the support graphs have been computed,

each iteration of the gEM(IO) algorithm may be applied on the support graph of Fig. 11 using the following steps (Sato and Kameya, 2001).

Algorithm 15: `Extract-CYK`

for $l = 1to|\Omega|$ **do**
  Initialize all $\varphi(\tau)$ to $\emptyset$ and all $Visited[]$ to NO;
  ClearStack(U);
  `Visit-CYK`($l, Start, 0, L$);
  for *k=1 to* $|U|$ **do**
    τ_k := PopStack(U);
  $\delta_l := < \tau_1, ..., \tau_{|U|} >$;

Algorithm 16: `Visit-CYK` (l, A, i, j)

Put τ=A(i,j);
$Visited[\tau] := YES$;
if $j = i + 1$ **then**
  if $A(i,j) \rightarrow w_j^l \in chart(i,j)$ **then**
    add a set $\{A \rightarrow w_j\}$ to $\varphi\tau$;
else
  foreach $A(i,j) \rightarrow B(i,k)C(i,k) \in chart(i,j)$ **do**
    add to $\varphi(\tau)$ a set $A \rightarrow BC, B(i,k), C(k,j)$;
    if *Visited[B(i,k)]=NO* **then**
      `Visit-CYK`(l, B, i, k);
    if *Visited[C(k,j)]=NO* **then**
      `Visit-CYK`(l, C, k, j);
  PushStack(τ, U);

a. Compute the inside (α) and explanation (α_m) probabilities for each support graph in a bottom-up fashion:

Initialization:

$$\alpha(i, i+1|A) = \theta(A \rightarrow w_{i+1}) \tag{3.23}$$

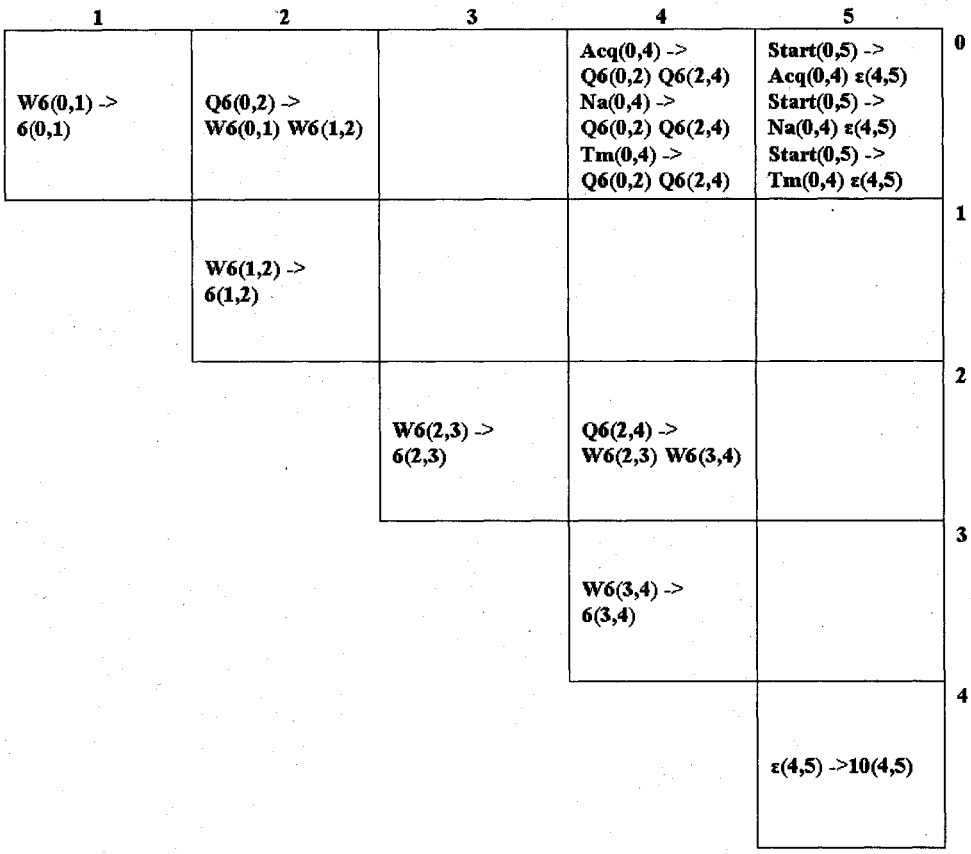


Figure 10 CYK tabular chart for example of Fig. 8.

Iterative process:

$$\begin{aligned} \alpha_m(i,j|A) &= \theta(A \rightarrow BC)\alpha(i,k|B)\alpha(k,j|C) \\ \alpha(i,j|A) &= \sum_m \alpha_m(i,j|A) \end{aligned} \tag{3.24}$$

where the summation extends over the branches of the support graph and the m^{th} branch represents a production rule of the form $A(i,j) \rightarrow B(i,k)C(k,j)$;

b. Compute the outside (β) probabilities and the balanced frequency of the rules (η) for each support graph in a top-down fashion:

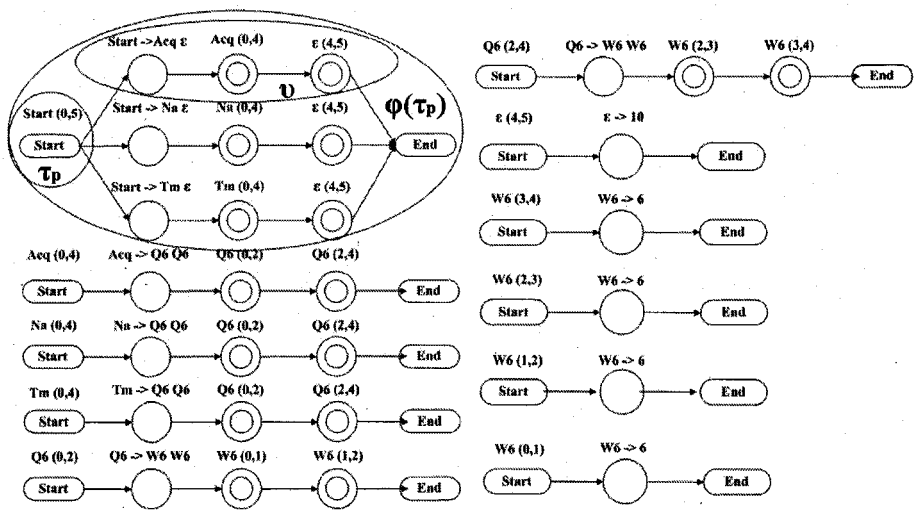


Figure 11 Support graphs for example of Fig. 8.

Initialization:

$$\begin{aligned} &\text{for } 0 \leq i < j \leq L,\ \beta(i,j|B) = 0, \quad \text{and} \quad \beta(0,L|Start) = 1 \\ &\text{and } \eta(A \rightarrow BC) = 0, \qquad A,B,C \in N \end{aligned} \tag{3.25}$$

Iterative process:

$$\begin{aligned} \beta(i,k|B) &\Leftarrow \beta(i,k|B) + \frac{\beta(i,j|A)\alpha_m(i,j|A)}{\alpha(i,k|B)} \\ \eta(A \rightarrow BC) &\Leftarrow \eta(A \rightarrow BC) + \frac{\beta(i,j|A)\alpha_m(i,j|A)}{\alpha(0,L|Start)} \\ \eta(A \rightarrow a) &\Leftarrow \eta(A \rightarrow a) + \frac{\beta(i,j|A)\alpha_m(i,j|A)}{\alpha(0,L|Start)} \end{aligned} \tag{3.26}$$

where L is the size of the sequence and the m^{th} branch of the support graph represents the production rule $A(i,j) \rightarrow B(i,k)C(k,j)$. It has been shown in (Sato *et al.*, 2001; Sato and Kameya, 2001) that these inside and outside probabilities cor-

respond to IO's, but are restrictions of Eq. 3.3 and 3.4 to the relevant contributing values from the support graphs;

c. Re-estimate the probabilities:

$$\theta'(A \rightarrow BC) = \frac{\eta(A \rightarrow BC)}{\sum_\lambda \eta(A \rightarrow \lambda)} \quad ; \quad \theta'(A \rightarrow a) = \frac{\eta(A \rightarrow a)}{\sum_\lambda \eta(A \rightarrow \lambda)} \tag{3.27}$$

For reference the inside and explanation probabilities are computed using the routine `Get-Inside-Probs(IO)` presented in Algorithm 18. Algorithm19 presents the routine `Get-Expectations(IO)` to compute the outside probabilities and η, and Algorithm 17 re-estimates the production probabilities.

The relation between the elements of these three steps and those of Eq. 3.2 can be expressed as follows.

$$\eta(A \rightarrow \lambda) = \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s) \tag{3.28}$$

Algorithm 17: `Graphical-EM(IO)`

```
Get-Inside-Probs();
loglikelihood(0) = Σ_{x∈Ω} α_x(0, L(x)|Start);
while loglikelihood(m)-loglikelihood(m − 1) > ε do
    Get-Expectation();
    foreach (A → BC) ∈ R do
        θ'(A → λ) = η(A → λ)/ Σ_{λ'} η(A → λ');
    m = m + 1;
    Get-Inside-Probs();
    loglikelihood(m) = Σ_{x∈Ω} α_x(0, L(x)|Start);
```

Algorithm 18: `Get-Inside-Probs()`

for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  for $k = |\delta_l|$ *to 1* **do**
    foreach $E \in \varphi(\tau_k)$ **do**
      $\alpha_E^l(\tau_k) = 1$;
      foreach $e \in E$ **do**
        if $e = (A \rightarrow \lambda)$ **then**
          $\alpha_E^l(\tau_k) = \alpha_E^l(\tau_k) \cdot \theta(A \rightarrow \lambda)$;
        else
          $\alpha_E^l(\tau_k) = \alpha_E^l(\tau_k) \cdot \alpha^l(e)$;
    $\alpha^l(\tau_k) = \sum_{E \in \varphi(\tau_k)} \alpha_E^l(\tau_k)$;

Algorithm 19: `Get-Expectations()`

foreach $(A \rightarrow \lambda) \in \alpha$ **do**
  $\eta(A \rightarrow \lambda) = 0$;
for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  $\beta^l(\tau_1) := 1$;
  for *k=2 to* $|\delta_l|$ **do**
    $\beta^l(\tau_k) := 0$;
  for *k=1 to* $|\delta_l|$ **do**
    foreach $E \in \varphi(\tau_k)$ **do**
      foreach $e \in E$ **do**
        if $e = (A \rightarrow \lambda)$ **then**
          $\eta(A \rightarrow \lambda) = \eta(A \rightarrow \lambda) + \beta^l(\tau_k) \cdot \alpha_E^l(\tau_k)/\alpha^l(0, L|Start)$;
        else
          if $\alpha^l(e) > 0$ **then**
            $\beta^l(e) = \beta^l(e) + \beta^l(\tau_k) \cdot \alpha_E^l(\tau_k)/\alpha_E^l(e)$;

Consider application of gEM to example of Fig. 8, using the production probabilities of Eq. 3.7. Suppose that the quantities $\alpha(0, 4|\texttt{Acq}) = 0.1667$, $\alpha(0, 4|\texttt{Na}) = 0.15$, $\alpha(0, 4|\texttt{Tm}) = 0.1408$ and $\alpha(4, 5|\epsilon) = 1$ have already been computed. Then, according to Eq. 3.24 $\alpha(0, 5|Start)$ will be computed as follows. Let $\alpha_1(0, 5|Start)$, $\alpha_2(0, 5|Start)$ and $\alpha_3(0, 5|Start)$ represent the explanation probabilities of the three branches in the first

support graph in Gig. 11. These can be computed as follows:

$$\begin{aligned}
\alpha_1(0,5|Start) &= \theta(Start \rightarrow \texttt{Acq}\ \epsilon)\alpha(0,4|\texttt{Acq})\alpha(4,5|\epsilon) \\
&= 0.1 \cdot 0.1667 \cdot 1 = 0.01667 \\
\alpha_2(0,5|Start) &= \theta(Start \rightarrow \texttt{Na}\ \epsilon)\alpha(0,4|\texttt{Na})\alpha(4,5|\epsilon) \\
&= 0.1 \cdot 0.5 \cdot 1 = 0.05 \\
\alpha_3(0,5|Start) &= \theta(Start \rightarrow \texttt{Tm}\ \epsilon)\alpha(0,4|\texttt{Tm})\alpha(4,5|\epsilon) \\
&= 0.1 \cdot 0.1408 \cdot 1 = 0.01408
\end{aligned} \tag{3.29}$$

Then $\alpha(0,5|Start)$ will be computed as in Eq. 3.30.

$$\begin{aligned}
\alpha(0,5|Start) &= \alpha_1(0,5|Start) + \alpha_2(0,5|Start) + \alpha_3(0,5|Start) \\
&= 0.01667 + 0.05 + 0.01408 \\
&= 0.08075
\end{aligned} \tag{3.30}$$

Moreover according to Eq. 3.26 $\eta(Start \rightarrow \texttt{Acq}\ \epsilon)$, $\eta(Start \rightarrow \texttt{Na}\ \epsilon)$ and $\eta(Start \rightarrow \texttt{Tm}\ \epsilon)$ will be computed as follows:

$$\begin{aligned}
\eta(Start \rightarrow \texttt{Acq}\ \epsilon) &= \frac{\beta(0,5|Start)\alpha_1(0,5|Start)}{\alpha(0,5|Start)} \\
&= \frac{1 \cdot 0.01667}{0.08075} = 0.20644
\end{aligned}$$

$$\begin{aligned}
\eta(Start \rightarrow \texttt{Na}\ \epsilon) &= \frac{\beta(0,5|Start)\alpha_2(0,5|Start)}{\alpha(0,5|Start)} \\
&= \frac{1 \cdot 0.05}{0.08075} = 0.619195
\end{aligned}$$

$$\eta(Start \rightarrow \texttt{Tm}\ \epsilon) = \frac{\beta(0,5|Start)\alpha_3(0,5|Start)}{\alpha(0,5|Start)}$$

$$= \frac{1 \cdot 0.01408}{0.08075} = 0.174365 \tag{3.31}$$

The reestimation formula for $\theta(Start \rightarrow \texttt{Acq}\ \epsilon)$ gives:

$$\begin{aligned} \theta(Start \rightarrow \texttt{Acq}\ \epsilon) &= \frac{\eta(Start \rightarrow \texttt{Acq}\ \epsilon)}{\eta(Start \rightarrow \texttt{Na}\ \epsilon) + \eta(Start \rightarrow \texttt{Acq}\ \epsilon) + \eta(Start \rightarrow \texttt{Tm}\ \epsilon)} \\ &= \frac{0.20644}{0.20644 + 0.619195 + 0.174365} = 0.20644 \end{aligned} \tag{3.32}$$

It has been shown that the results produced by the graphical EM are the same as the results given by the IO algorithm (Sato *et al.*, 2001; Sato and Kameya, 2001). Actually, while the IO algorithm passes through all the possible combinations of a grammar to produce a sequence, the graphical EM algorithm only uses the combinations given by the support graphs. It is more efficient in most practical cases, although the worst case time complexity per iteration is equal to IO's. A greater memory complexity of $O(M_{nt}^3 \cdot L^2)$ is however needed to store the support graphs.

3.2.3 A VS version of gEM

Since VS has, in practice, a lower time complexity per iteration and usually converges faster that IO, a VS version of gEM(IO) has been proposed, leading to a new algorithm called gEM(VS) (Latombe *et al.*, 2006e). This algorithm modifies the support graphs in order to access the *best* derivation of the corresponding sequence, resulting in a set of one-branch support graphs. `Get-InsideProbs(VS)` of Algorithm 18 now computes the maximum probabilities instead of the inside probabilities, but still computes the explanation probabilities of the remaining parts of the support graphs. Fig. 12 shows a

possible result of the support graphs corresponding to gEM(VS) from the same example as in Fig. 11. Then, the routine `Get-Expectations(VS)` computes the outside probabilities and η based only on the new set of support graphs.

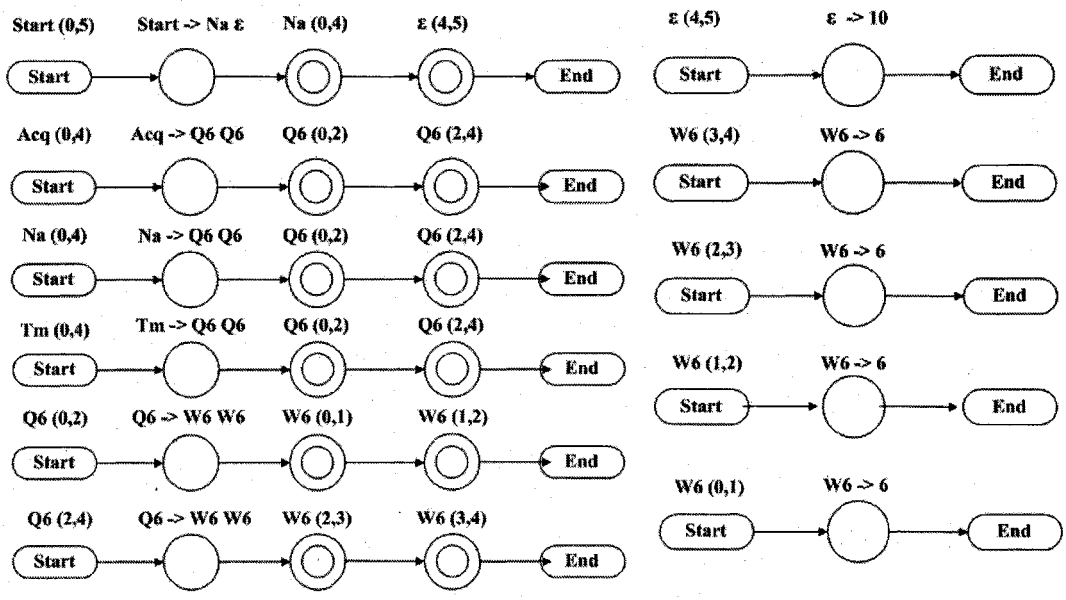


Figure 12 New support graphs for gEM(VS) based on Fig. 11.

Once the original support graphs have been precomputed, an iteration of the gEM(VS) algorithm may be applied on the support graph of Fig. 11 using the following steps:

a. Compute the maximum ($\hat{\alpha}$) and explanation ($\hat{\alpha}_m$) probabilities for each support graph in a bottom-up fashion:

Initialization:

$$\hat{\alpha}(i, i+1|A) = \theta(A \rightarrow w_{i+1}) \tag{3.33}$$

Iterative process:

$$\begin{aligned} \hat{\alpha_m}(i,j|A) &= \theta(A \rightarrow BC)\hat{\alpha}(i,k|B)\hat{\alpha}(k,j|C) \\ \hat{\alpha}(i,j|A) &= \max_m\{\hat{\alpha_m}(i,j|A)\} \end{aligned} \tag{3.34}$$

where the maximization extends over the branches of the support graphs and the m^{th} branch represents a production rule of the form $A(i,j) \rightarrow B(i,k)C(k,j)$;

b. Compute the outside ($\hat{\beta}$) probabilities and the balanced frequency of the rules ($\hat{\eta}$) for each support graph in a top-down fashion:

Initialization:

$$\begin{aligned} &\text{for } 0 \leq i < j \leq L,\ \hat{\beta}(i,j|B) = 0, \quad \text{and} \quad \hat{\beta}(0,L|Start) = 1 \\ &\qquad\qquad \text{and} \quad \hat{\eta}(A \rightarrow BC) = 0, \qquad A,B,C \in N \end{aligned} \tag{3.35}$$

Iterative process:

$$\text{If } \hat{\beta}(i,k|A) \neq 0: \qquad \begin{aligned} \hat{\beta}(i,k|B) &= \frac{\hat{\beta}(i,j|A)\hat{\alpha_m}(i,j|A)}{\hat{\alpha}(i,k|B)} \\ \hat{\eta}(A \rightarrow BC) &= \frac{\hat{\beta}(i,j|A)\hat{\alpha_m}(i,j|A)}{\hat{\alpha}(0,L|Start)} \\ \hat{\eta}(A \rightarrow a) &= \frac{\hat{\beta}(i,j|A)\hat{\alpha_m}(i,j|A)}{\hat{\alpha}(0,L|Start)} \end{aligned} \tag{3.36}$$

Here, m identifies the most probable branch of the support graph as determined by Eq. 3.34, and corresponds to the production rule $A(i,j) \rightarrow B(i,k)C(k,j)$. L is still the size of the sequence. In the computation of $\hat{\eta}$, the normalization is always performed with respect to $\hat{\alpha}(0,L|Start)$, whatever the corresponding rule;

c. Reestimate the probabilities:

$$\theta'(A \rightarrow BC) = \frac{\hat{\eta}(A \rightarrow BC)}{\sum_{\lambda} \hat{\eta}(A \rightarrow \lambda)} \quad ; \quad \theta'(A \rightarrow a) = \frac{\hat{\eta}(A \rightarrow a)}{\sum_{\lambda} \hat{\eta}(A \rightarrow \lambda)} \tag{3.37}$$

These steps are performed using Algorithms 20 through 22.

The relation between the elements of these steps and those of Eq. 3.2 can be expressed as follows.

$$\hat{\eta}(A \rightarrow \lambda) = \sum_{x \in \Omega} N(A \rightarrow \lambda, \hat{d}_x) \tag{3.38}$$

Algorithm 20: `Graphical-EM(VS)`

```
Get-Inside-Probs-VS();
loglikelihood(0) = Σ_{x∈|Ω|} α_x(0, L(x)|Start);
while loglikelihood(m)-loglikelihood(m − 1) > ε do
    Get-Expectation-VS();
    foreach (A → BC) ∈ R do
        θ'(A → λ) = η̂(A → λ)/ Σ_{λ'} η̂(A → λ');
    m = m + 1;
    Get-Inside-Probs-VS();
    loglikelihood(m) = Σ_{x∈|Ω|} α_x(0, L(x)|Start);
```

The support graphs of the example of Section 3.2.2 obtained by using the Viterbi version of the algorithm are given in Fig. 12. Suppose that the production rule probabilities have been initialized like in Eq. 3.7, and that the quantities $\hat{\alpha}(0, 4|\texttt{Acq}) = 0.1667$, $\hat{\alpha}(0, 4|\texttt{Na}) = 0.5$, $\hat{\alpha}(0, 4|\texttt{Tm}) = 0.1408$ and $\hat{\alpha}(4, 5|\epsilon) = 1$ have already been computed. Then, according to Eq. 3.34 $\hat{\alpha}(0, 5|Start)$ will be computed as follows:

$$\begin{aligned}
\hat{\alpha}(0, 5|Start) &= \max_{m}\{\hat{\alpha}_m(0, 5|Start)\} \quad . \\
&= \max\{\theta(Start \rightarrow Acq\epsilon)\hat{\alpha}(0, 4|\texttt{Acq})\hat{\alpha}(4, 5|\epsilon), \theta(Start \rightarrow Na\epsilon)\hat{\alpha}(0, 4|\texttt{Na}))\hat{\alpha}(4, 5 \\
&= \theta(Start \rightarrow Na\epsilon)\hat{\alpha}(0, 4|\texttt{Na}))\hat{\alpha}(4, 5|\epsilon) \\
&= 0.05
\end{aligned}$$

Algorithm 21: `Get-Inside-Probs-VS()`

for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  for $k = |\delta_l|$ *to 1* **do**
    foreach $E \in \varphi(\tau_k)$ **do**
      $\hat{\alpha}^l_E(\tau_k) = 1$;
      foreach $e \in E$ **do**
        if $e = (A \rightarrow \lambda)$ **then**
          $\hat{\alpha}^l_E(\tau_k) = \hat{\alpha}^l_E(\tau_k) \cdot \theta(A \rightarrow \lambda)$;
        else
          $\hat{\alpha}^l_E(\tau_k) = \hat{\alpha}^l_E(\tau_k) \cdot \alpha[l, e]$;
    $\hat{\alpha}^l(\tau_k) = \max_{E \in \varphi(\tau_k)}\{\hat{\alpha}^l_E(\tau_k)\}$;
    $E_{max} = \text{argmax}_{E \in \varphi(\tau_k)}\{\hat{\alpha}^l_E(\tau_k)\}$;
    $\psi^l_{viterbi}(\tau_k) = \varphi^l(\tau_k)(E_{max})$;

Algorithm 22: `Get-Expectations-VS()`

foreach $(A \rightarrow \lambda) \in R$ **do**
  $\eta[A \rightarrow \zeta] = 0$;
for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  $\hat{\beta}^l(\tau_1) := 1$;
  for *k=2 to* $|\delta_l|$ **do**
    $\hat{\beta}^l(\tau_k) := 0$;
  for *k=1 to* $|\delta_l|$ **do**
    foreach $e \in \varphi_{viterbi}(\tau_k)$ **do**
      if $e = (A \rightarrow \lambda)$ **then**
        $\hat{\eta}(A \rightarrow \lambda) = \hat{\eta}(A \rightarrow \lambda) + \hat{\beta}^l(\tau_k) \cdot \hat{\alpha}^l_E(\tau_k)/\hat{\alpha}^l(S(0, n_l))$;
      else
        if $\hat{\alpha}^l(e) > 0$ **then**
          $\hat{\beta}^l(e) = \hat{\beta}^l(e) + \hat{\beta}^l(\tau_k) \cdot \hat{\alpha}^l_E(\tau_k)/\hat{\alpha}^l(e)$;

Moreover according to Eq. 3.36 $\hat{\eta}(Start \rightarrow \texttt{Acq}\ \epsilon)$, $\hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon)$ and $\hat{\eta}(Start \rightarrow \texttt{Tm}\ \epsilon)$ will be computed as follows:

$$\hat{\eta}(Start \rightarrow \texttt{Acq}\ \epsilon) = 0$$

$$\begin{aligned}\hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon) &= \frac{\hat{\beta}(0,5|Start)\hat{\alpha}_2(0,5|Start)}{\hat{\alpha}(0,5|Start)} = \frac{1 \cdot 0.05}{0.05} = 1 \\ \hat{\eta}(Start \rightarrow \texttt{Tm}\ \epsilon) &= 0 \end{aligned} \tag{3.40}$$

The reestimation formula for $\theta'(Start \rightarrow \texttt{Acq}\ \epsilon)$ gives:

$$\begin{aligned}\theta'(Start \rightarrow \texttt{Na}\ \epsilon) &= \frac{\hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon)}{\hat{\eta}(Start \rightarrow \texttt{Acq}\ \epsilon) + \hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon) + \hat{\eta}(Start \rightarrow \texttt{Tm}\ \epsilon)} \\ &= \frac{1}{0+1+0} = 1 \end{aligned} \tag{3.41}$$

And the probabilities have to be normalized as in Eq. 3.37.

After the first step of the iterative process, all the support graphs are kept. Indeed, as `Get-Inside-Probs(VS)` runs in a bottom-up fashion, support graphs coming from a deleted branch located higher in the support graphs will be present at the end of the routine. As `Get-Expectations(VS)` works only on the new support graph in a top-down fashion, and thanks to the initialization of $\hat{\beta}$, the useless support graphs will not have effect on the computation of $\hat{\eta}$. Indeed $\hat{\eta}$ is computed by considering only the best derivation tree. Since the second pass must be performed on these one-branch support graphs, gEM(VS) requires more memory than gEM(IO) for storage. However, both gEM(VS) and gEM(IO) have time and memory complexities that grow with $O(M_{nt}^3 \cdot L^3)$ and $O(M_{nt}^3 \cdot L^2)$.

3.3 Fast gradient descent based on relative entropy – HOLA

Oates and Heeringa (2002) present a heuristic on-line algorithm called HOLA based on summary statistics. Unlike IO, VS, TS or gEM, HOLA does not optimize the likelihood of the training dataset in a EM fashion to re-compute production rule probabilities. Instead, it re-estimates the probabilities of a grammar by using an approximation of the gradient descent. It requires pre-processing prior to reestimation of the probabilities. A routine

computes summary statistics for the training database. A standard Earley (Earley, 1970) or CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) chart parser is used to count how many times a particular rule is used in the production of the sequences of the training database. For a grammar under CNF the amount of data needed is two matrices, one for the transition rules and a second for the emission rules. These matrices are stored, and can then be updated if new data is added. The values of the frequencies of the rules are then normalized according to Eq. 3.42:

$$N'(A \rightarrow \lambda).o = \frac{N(A \rightarrow \lambda).o}{N(A).o} \tag{3.42}$$

where $N(A \rightarrow \lambda).o$ represents the frequency of appearance of the rule $A \rightarrow \lambda$ when parsing the training dataset, and $N(A).o$ represents the frequency of A being the origin of any production rule. This allows a comparison of counts with different numbers of sequences. For example of Fig. 10, the $N().o$ will store the following values:

$$\begin{aligned}
&N(Start \rightarrow \texttt{Acq}\ \epsilon).o = 1 \quad && N(Start \rightarrow \texttt{Na}\ \epsilon).o = 1 \\
&N(Start \rightarrow \texttt{Tm}\ \epsilon).o = 1 \quad && N(\texttt{Acq} \rightarrow Q6\ Q6).o = 1 \\
&N(\texttt{Na} \rightarrow Q6\ Q6).o = 1 \quad && N(\texttt{Tm} \rightarrow Q6\ Q6).o = 1 \\
&N(Q6 \rightarrow W6\ W6).o = 2 \quad && N(W6 \rightarrow 6).o = 4 \\
&N(\epsilon \rightarrow 10).o = 1
\end{aligned} \tag{3.43}$$

Once the histograms have been computed, an iteration of the HOLA algorithm may be applied using the following steps, as presented in Algorithm 25:

a. Generate sequences from the SCFG G_s: Starting from the $Start$ symbol, randomly generate a particular set of derivation trees that lead to a corresponding set of sequences, using the probabilities of G_s. In other words, the sequences depend on the current distribution of the probabilities of G_s;

b. Compute and normalize the corresponding frequency histograms: Parse the generated sequences and count the total number of times $N(A \rightarrow \lambda).s$ that the production rule $A \rightarrow \lambda$ occurs in the complete set of CYK charts. Then normalize to compute $N'(A \rightarrow \lambda).s$;

c. Re-estimate the probabilities:

$$\theta'(A \rightarrow \lambda) = \theta(A \rightarrow \lambda)(1 + \chi(N'(A \rightarrow \lambda).o - N'(A \rightarrow \lambda).s)) \tag{3.44}$$

Note that $\theta'(A \rightarrow \lambda)$ must be subsequently normalized to remain consistent.

The pseudo-codes of the algorithms are given below in Algorithms 23 through 25.

Algorithm 23: `HOLA()`

```
load HOLA-Count-Storage-Normalized;
while criterion is not reached do
    HOLA-Iteration;
```

Algorithm 24: `HOLA-Count()`

```
derivation = parse each sequence of the database;
foreach d ∈ derivation do
    foreach rule r in the grammar do
        if r = d then
            N(r).o = N(r).o + 1;
save HOLA-Count-Storage-Raw;
{N'(r).o}=normalize {N(r).o};
save HOLA-Count-Storage-Normalized;
```

Algorithm 25: `HOLA-Iteration()`

generate a set of sequences according to the grammar;
derivation = parse each generated sequence;
foreach $d \in$ *derivation* **do**
    foreach *rule* r *in the grammar* **do**
        if $r = d$ **then**
            $N(r).s = N(r).s + 1$;
$\{N'(r).s\}$=normalize $\{N(r).s\}$;
foreach *rule* r **do**
    $\theta(r) = \theta(r) \cdot (1 + \chi \cdot (N'(r).o - N'(r).s))$;
    if $\theta(r) \leq 0$ **then**
        $\theta(r) = min$;

It can be seen that three parameters must be set with HOLA – the number of sequences to randomly generate at each iteration, the learning rate χ in the reestimation formula, and the stopping criterion. Parameter values depend on the application, and will vary according to the average size of the sequences, the size of the grammar, etc. As the purpose of HOLA is to have the number $N'(A \rightarrow \lambda).s$ tend toward $N'(A \rightarrow \lambda).o$ for every rule $A \rightarrow \lambda$, a possible stopping criterion consists in making the relative entropy converge on an independent validation set. The relative entropy of two distributions p and q is given by:

$$H = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{3.45}$$

In this case, for a grammar under CNF, p and q are the sample and the observed distribution of the frequency of a non-terminal producing a combination of two non-terminals or producing a terminal. The sum of the relative entropies over all the non-terminals becomes:

$$HH = \sum_{A \rightarrow \lambda} N'(A \rightarrow \lambda).s \log \frac{N'(A \rightarrow \lambda).s}{N'(A \rightarrow \lambda).o} \tag{3.46}$$

Here the sum is extended over all production rules $A \rightarrow \lambda$ in the grammar. This is used in the convergence criterion:

$$|(HH(iteration) - HH(iteration - 1)| < \varepsilon \tag{3.47}$$

Suppose the database is composed by only the sequence of Fig. 8. Then the values of the raw and normalized observed histograms (N and N') will be:

$$\begin{aligned}
&N(Start \rightarrow \texttt{Acq}\ \epsilon).o = 1\\
&N(Start \rightarrow \texttt{Na}\ \epsilon).o = 1\\
&N(Start \rightarrow \texttt{Tm}\ \epsilon).o = 1\\
&N(\texttt{Acq} \rightarrow Q6\ Q6).o = 1\\
&N(\texttt{Na} \rightarrow Q6\ Q6).o = 1\\
&N(\texttt{Tm} \rightarrow Q6\ Q6).o = 1\\
&N(Q6 \rightarrow W6\ W6).o = 2\\
&N(W6 \rightarrow 6).o = 4\\
&N(\epsilon \rightarrow 10).o = 4
\end{aligned} \tag{3.48}$$

and

$$\begin{aligned}
&N'(Start \rightarrow \texttt{Acq}\ \epsilon).o = 1/3\\
&N'(Start \rightarrow \texttt{Na}\ \epsilon).o = 1/3\\
&N'(Start \rightarrow \texttt{Tm}\ \epsilon).o = 1/3\\
&N'(\texttt{Acq} \rightarrow Q6\ Q6).o = 1\\
&N'(\texttt{Na} \rightarrow Q6\ Q6).o = 1\\
&N'(\texttt{Tm} \rightarrow Q6\ Q6).o = 1
\end{aligned}$$

$$N'(Q6 \rightarrow W6\ W6).o = 1$$
$$N'(W6 \rightarrow 6).o = 1$$
$$N'(\epsilon \rightarrow 10).o = 1 \tag{3.49}$$

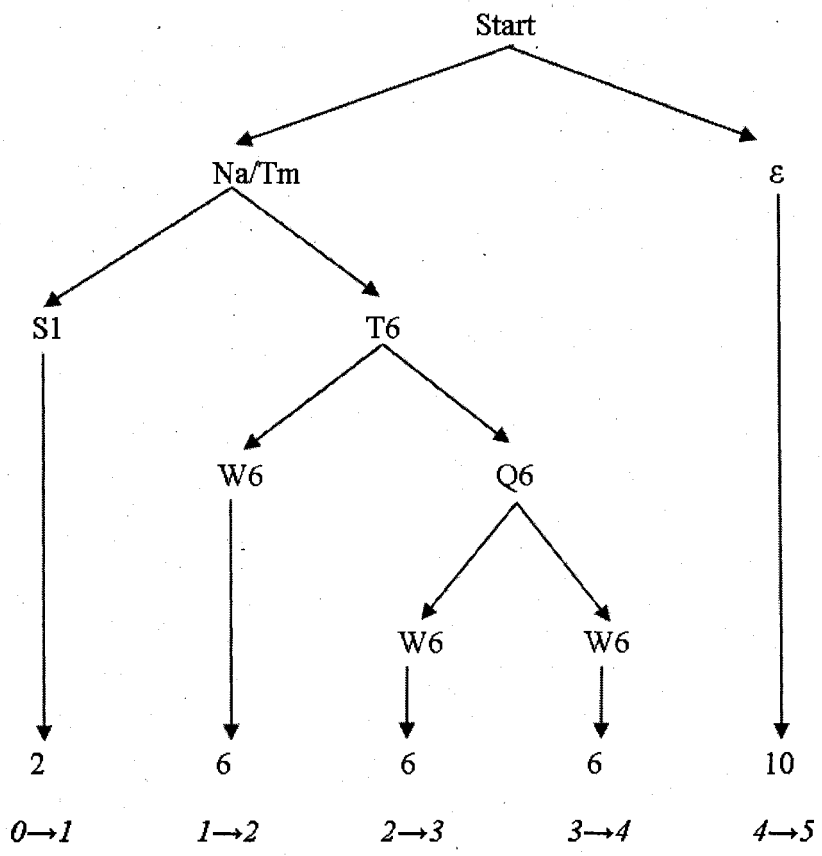


Figure 13 Example of derivation tree and sequence generated by the grammar for Mercury.

All the other rule frequencies are set to 0. Suppose the grammar whose probabilities are presented below generates the sequence of Fig. 13. Note that some production probabilities are not involved in this example, and that for consistency, $\sum_\lambda \theta(A \rightarrow \lambda) = 1$.

$$\theta(Start \rightarrow \texttt{Acq}\ \epsilon) = 0.1$$
$$\theta(Start \rightarrow \texttt{Na}\ \epsilon) = 0.1$$
$$\theta(Start \rightarrow \texttt{Tm}\ \epsilon) = 0.1$$
$$\theta(\texttt{Acq} \rightarrow Q6\ Q6) = 0.1667$$
$$\theta(\texttt{Na} \rightarrow Q6\ Q6) = 0.5$$
$$\theta(\texttt{Tm} \rightarrow Q6\ Q6) = 0.1408$$
$$\theta(\texttt{Na} \rightarrow S1\ T6) = 0.5$$
$$\theta(\texttt{Tm} \rightarrow S1\ T6) = 0.1408$$
$$\theta(T6 \rightarrow W6\ Q6) = 1$$
$$\theta(Q6 \rightarrow W6\ W6) = 1$$
$$\theta(S1 \rightarrow 2) = 0.2$$
$$\theta(W6 \rightarrow 6) = 1$$
$$\theta(\epsilon \rightarrow 10) = 1 \qquad (3.50)$$

The values of the raw and normalized sample histograms (N and N') for this sequence will be:

$$N(Start \rightarrow \texttt{Acq}\ \epsilon).s = 0$$
$$N(Start \rightarrow \texttt{Na}\ \epsilon).s = 1$$
$$N(Start \rightarrow \texttt{Tm}\ \epsilon).s = 1$$
$$N(\texttt{Acq} \rightarrow Q6\ Q6).s = 0$$
$$N(\texttt{Na} \rightarrow Q6\ Q6).s = 0$$
$$N(\texttt{Tm} \rightarrow Q6\ Q6).s = 0$$
$$N(\texttt{Na} \rightarrow S1\ T6).s = 1$$

$$
\begin{aligned}
&N(\texttt{Tm} \rightarrow S1\ T6).s = 1 \\
&N(T6 \rightarrow W6\ Q6).s = 1 \\
&N(Q6 \rightarrow W6\ W6).s = 1 \\
&N(S1 \rightarrow 2).s = 1 \\
&N(W6 \rightarrow 6).s = 3 \\
&N(\epsilon \rightarrow 10).s = 1
\end{aligned} \tag{3.51}
$$

and

$$
\begin{aligned}
&N'(Start \rightarrow \texttt{Acq}\ \epsilon).s = 0 \\
&N'(Start \rightarrow \texttt{Na}\ \epsilon).s = 1/2 \\
&N'(Start \rightarrow \texttt{Tm}\ \epsilon).s = 1/2 \\
&N'(\texttt{Acq} \rightarrow Q6\ Q6).s = 0 \\
&N'(\texttt{Na} \rightarrow Q6\ Q6).s = 0 \\
&N'(\texttt{Tm} \rightarrow Q6\ Q6).s = 0 \\
&N'(\texttt{Na} \rightarrow S1\ T6).s = 1 \\
&N'(\texttt{Tm} \rightarrow S1\ T6).s = 1 \\
&N'(T6 \rightarrow W6\ Q6).s = 1 \\
&N'(Q6 \rightarrow W6\ W6).s = 1 \\
&N'(S1 \rightarrow 2).s = 1 \\
&N'(W6 \rightarrow 6).s = 1 \\
&N'(\epsilon \rightarrow 10).s = 1
\end{aligned} \tag{3.52}
$$

Thus, setting χ to 0.1, the probabilities are re-estimated using Eq. 3.44 as follows:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = 0.1 \cdot (1 + 0.1 \cdot (1/3 - 0)) = 0.103$$

$$\theta'(Start \rightarrow \texttt{Na}\ \epsilon) = 0.1 \cdot (1 + 0.1 \cdot (1/3 - 1/2)) = 0.098$$
$$\theta'(Start \rightarrow \texttt{Tm}\ \epsilon) = 0.1 \cdot (1 + 0.1 \cdot (1/3 - 1/2)) = 0.098$$
$$\theta'(\texttt{Acq} \rightarrow Q6\ Q6) = 0.1667 \cdot (1 + 0.1 \cdot (1 - 0)) = 0.18337$$
$$\theta'(\texttt{Na} \rightarrow Q6\ Q6) = 0.5 \cdot (1 + 0.1 \cdot (1 - 0)) = 0.55$$
$$\theta'(\texttt{Tm} \rightarrow Q6\ Q6) = 0.1408 \cdot (1 + 0.1 \cdot (1 - 0)) = 0.15488$$
$$\theta'(\texttt{Na} \rightarrow S1\ T6) = 0.5 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.45$$
$$\theta'(\texttt{Tm} \rightarrow S1\ T6) = 0.1408 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.12672$$
$$\theta'(T6 \rightarrow W6\ Q6) = 1 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.9$$
$$\theta'(Q6 \rightarrow W6\ W6) = 1 \cdot (1 + 0.1 \cdot (1 - 1)) = 1$$
$$\theta'(S1 \rightarrow 2) = 0.2 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.18$$
$$\theta'(W6 \rightarrow 6) = 1 \cdot (1 + 0.1 \cdot (1 - 1)) = 1$$
$$\theta'(\epsilon \rightarrow 6) = 1 \cdot (1 + 0.1 \cdot (1 - 1)) = 1 \tag{3.53}$$

The probabilities should thereafter be normalized over all the production probabilities, including those which are not involved in the example, to be consistent.

HOLA's time complexity per iteration only depends on the size of the histograms and is constant with regard to the amount of data. HOLA also has a very low memory complexity. This algorithm has a time complexity and a memory complexity that are both of $O(M_{nt}^3)$. On the other hand, the number of iterations will vary with regard to parameters, such as the number of sequences randomly generated, the value of χ and the initial distribution of the production probabilities.

3.4 Comparison of learning techniques

It has already been mentioned that gEM and TS were numerically equivalent. Moreover, gEM(IO) and TS(IO) are numerically equivalent to the classical IO algorithm, while

gEM(VS) and TS(VS) are numerically equivalent to the classical VS algorithm. The differences between these two techniques lie in both pre-processing and iterative procedures.

Pre-processing with TS results in a non-compact representation of the derivation trees of the sequences of the training dataset. This allows for very fast re-estimation of the data during the iterative process. However, the derivation trees can be listed only if the grammar is low ambiguous, which limits the total number of derivation trees for a given sequence, otherwise memory requirements become an issue. Pre-processing with gEM creates support graphs, which is a more compact representation of the derivation trees, and thereby allows dealing with more ambiguous grammar, but resulting in a higher time complexity per iteration for low-ambiguity grammars.

HOLA is very different from the other algorithms, although it also uses a parser for pre-processing of the training dataset. Pre-processing in this case results in an histogram summarizing the frequency of appearance of the rules during the parsing, and therefore has a size that only depends from the number of non-terminals of the grammar, which is very low for radar ES applications. Its time complexity also depends only from the number of non-terminals of the grammar, and the iterative process is therfore much faster than those of TS and gEM. However, it has the main inconvenience of not optimizing the likelihood of the training dataset, which may lead to lower accuracy than TS and gEM.

Table I displays a summary of the re-estimation formulas of the above-described techniques, namely IO, VS, TS(IO), TS(VS), gEM(IO), gEM(VS) and HOLA, along with their relation with the global re-estimation formula of Eq.3.2. This table contains definitions of the symbols used in each reestimation formula. IO re-estimates the production rule probabilities using inside and outside probabilities, while VS does this using a simplification of Eq. 3.2. TS(IO) uses Eq. 3.2 directly, while TS (VS) uses the same equation as VS. gEM(IO) and gEM(VS) re-estimate the probabilities by normalizing the η and $\hat{\eta}$. Note,

however, that TS/gEM(IO) and TS/gEM(VS) all have equivalent formulas. In contrast, HOLA uses an approximation of the gradient descent.

Table I

Re-estimation formulas of techniques for batch learning of SCFGs.

Method	Reestimation Formula
General	$\theta'(A \to \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x \mid G_s)}{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x \mid G_s)}$ $N(A \to \lambda, d_x)$ = frequency of the rule $A \to \lambda$ in the derivation tree d_x
IO	$\theta'(A \to BC) = \frac{\sum_{x \in \Omega} \frac{\sum_{0 \le i < k < j \le L} \alpha(i,k \mid B)\alpha(k,j \mid C)\beta(i,j \mid A)\theta(A \to BC)}{\alpha(0,L \mid Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j \mid A)\beta(i,j \mid A)}{\alpha(0,L \mid Start)}}$ $\theta'(A \to a) = \frac{\sum_{x \in \Omega} \frac{\sum_{i \mid w_i = a} \beta(i-1,i \mid A)\theta(A \to a)}{\alpha(0,L \mid Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j \mid A)\beta(i,j \mid A)}{\alpha(0,L \mid Start)}}$ $\alpha(i,j \mid A)$=inside probability of A generating the subsequence $w_{i+1}, ..., w_j$ $\beta(i,j \mid A)$=outside probability of A generating the subsequence $w_{i+1}, ..., w_j$
VS	$\theta'(A \to \lambda) = \frac{N(A \to \lambda, \hat{d}_x)}{N(A, \hat{d}_x)}$ $\hat{d}_x$ = best derivation tree of sequence x
TS(IO)	$\theta'(A \to \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x \mid G_s)}{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x \mid G_s)}$

Method	Reestimation Formula
TS(VS)	$\theta'(A \to \lambda) = \frac{N(A \to \lambda, \bar{d}_x)}{N(A, \bar{d}_x)}$
gEM(IO)	$\theta'(A \to \lambda) = \frac{\eta(A \to \lambda)}{\sum_\lambda \eta(A \to \lambda)}$ $\eta(A \to \lambda)$=sum on all the strings of the normalized balanced frequency of the rule $A \to \lambda$
gEM(VS)	$\theta'(A \to \lambda) = \frac{\hat{\eta}(A \to \lambda)}{\sum_\lambda \hat{\eta}(A \to \lambda)}$ $\hat{\eta}(A \to \lambda)$=$N(A \to \lambda, \hat{d}_x)$
HOLA	$\theta'(A \to \lambda) = \theta(A \to \lambda) \cdot (1 + \chi \cdot (N(A \to \lambda).o - N(A \to \lambda).s))$ $N(A \to \lambda).o$ = frequency of the rule $A \to \lambda$ over the training set $N(A \to \lambda).s$ = frequency of the rule $A \to \lambda$ over the generated set

CHAPTER 4

INCREMENTAL DERIVATIONS OF GRAPHICAL EM

In radar ES applications, new information from a battlefield or other sources often becomes available in blocks at different times. Since timely protection against threats is often of vital importance, fast *incremental* learning of SCFG probabilities is an undisputed asset. In this context, incremental learning refers to the ability to adapt SCFG probabilities from new blocks of training sequences, without requiring access to training sequences used to learn the existing SCFG. Incremental learning must also preserve previously-acquired knowledge.

From a radar ES perspective, the IO and VS algorithm have the drawback of being computationally very demanding, and not allowing for incremental learning of SCFG probabilities. If new training data becomes available, it must be added with all previous training sequences, and used to re-train the production rule probabilities from the start. This process has a high overall time complexity. The fast alternatives TS and gEM are only semi-incremental: if new data is added, the results from the pre-processing can be stored incrementally, but a new iterative process has to be redone from scratch using all cumulative data resulting from pre-processing. Faster pre-processing allows reducing the overall time complexity when new data becomes available. Only HOLA is incremental. If new data is added, one just updates the histogram $N().o$, and re-starts the iterative process using the result from previous training. This should theoretically lead to small convergence time, since the production rule probabilities incorporate a priori information on the probability distributions.

Experimental results (presented in Sub-section 6.1) indicate that since HOLA optimizes the relative entropy between two distributions, it yields lower accuracy than TS or gEM.

In addition, TS should lead to very high time complexity per iteration for ambiguous grammars. Thus, in order to achieve the best trade-off in terms of accuracy and resources requirements, one may either try to design a more accurate HOLA algorithm, or to design an incremental gEM algorithm. In light of techniques found in the literature, this last option seems more feasible and is developed in this chapter.

In literature, several variants of the EM algorithm can provide inspiration for the development of an incremental gEM algorithm. Several EM-based algorithms for on-line or sequential optimization of parameters have been proposed for different applications. Neal and Hinton (1998) proposed an incremental version of the basic EM algorithm based on the fact that some parameters are reestimated using a vector of sufficient statistics. The incremental EM algorithm consists in computing the new sufficient statistics of a selected block of data in the E-step, and reestimating the parameters by combining the old sufficient statistics (of the unused dataset) and the new ones in the M-step. With this approach, one can choose to update the parameters by selecting data blocks cyclicly, or by giving preference to some scheme for which the algorithm has not converged. The authors have observed that this leads to a shorter convergence time.

Based on this general approach, several incremental algorithms have been developed in the literature, mainly to estimate HMM probabilities. For instance, Digalakis (1999) proposed an online EM algorithm that updates the parameters on a HMM after each sentence, with only one pass through the data. Gotoh *et al.* (1998) presented an incremental estimation approach for HMM parameters in order to accelerate the reestimation of the probabilities through a process selecting different blocks of data to update the parameters, until convergence is achieved.

Sato and Ishii (2000) proposes an on-line version of the EM algorithm for normalized Gaussian networks. It is based on an approximation of the weighted means of the variables with respect to the posterior probability, from its previous value. Then these approximated

weighted means are combined during the reestimation phase of the process to lead to the new estimation of the parameters.

A different reference approach to estimate parameters using incomplete data is proposed by Titterington (1984). This recursive approach is adapted to EM technique in the following way. A classical auxiliary function is computed during the E-step, and recursion is used to approximate the re-estimation of the parameter θ during the M-step. At iteration $k+1$, the recursion computes θ_{k+1} using (a) the previous value θ_k; (b) a Fisher matrix corresponding to a complete observation of the data; (c) a vector of scores. Chung and Bohme (2003) improves this technique by proposing an adaptive procedure to determine the step size at each recursion, to reduce the convergence time. Jorgensen (1999) investigates a dynamic form of EM, based on Titterington's algorithm, to reestimate mixture proportions that require a single EM update for each observation.

Finally, Baldi and Chauvin (1994) proposes a smooth on-line algorithm to learn the parameters of a HMM. It is not based on EM or Baum-Welch algorithms as with the majority of the existing algorithms, but on a simple gradient descent to minimize negative likelihood. To avoid non-positive values, production rule probabilities are expressed using an exponential form, upon which this algorithm is applied. Considering a given sample of data, the frequency of the rules in the derivation trees are computed, and the gradient is computed by normalizing the frequencies and subtracting them from the current value of the corresponding production rule. Probabilities are reestimated based on those of the previous iteration. The authors ensure that this algorithm can either be applied in a batch, sequential, or on-line fashion.

The rest of this chapter presents two versions of gEM, named incremental gEM (igEM) and online igEM (oigEM), that are based on research by Neal and Hinton (1998), and by Sato and Ishii (2000), respectively. This work has also been introduced in (Latombe *et al.*, 2006f) and (Latombe *et al.*, 2006d). These algorithms have the advantage of al-

lowing to adapt to incremental learning as desired in this thesis. Prior knowledge of all the training sequences is not required for training, such as with Titterington's algorithm. Moreover, the re-initialization of the production rule probabilities with igEM and oigEM may help avoid getting trapped in local optima (*cf.* Section 4.3). Techniques based on gradient descent starting from the previous SCFG probabilities may encounter difficulties during incremental learning, as they are likely to provide solutions that get trapped in local optima.

4.1 Incremental gEM

4.1.1 The Expectation-Maximization algorithm

In many real-world problems, it is impossible to obtain complete datasets, without any corrupted or even missing sample. Moreover, one often has access to data generated by a system whose corresponding state is unknown but of vital importance. The EM algorithm allows the estimation of a parameter θ by optimizing some chosen likelihood in problems depending on unobserved variables. It consists of two steps:

- **The *Expectation* step (E-step):** given a set of observed variables $x_1^n = \{x_1, ..., x_n\}$ and a set of unobserved ones $Z_1^n = \{Z_1, ..., Z_n\}$, it computes $P(Z_i|x_i, \theta)$ to get the objective function $Q(\theta, \theta^{(m)}) = E_{Z_1^n}[logP(x_1^n, Z_1^n|\theta)|x_1^n, \theta^{(m)}]$, where θ is the parameter to estimate and $\theta^{(m)}$ its estimation at iteration m;
- **The *Maximization* step (M-step):** compute $\theta^{(m+1)} = \text{argmax}_\theta\{Q(\theta, \theta^{(m)})\}$.

4.1.2 Incremental EM

Neal and Hinton (1998) propose a version of the EM algorithm based on sufficient statistics instead of the raw data. Given a sample $\{x_1, ..., x_n\}$ governed by the density function $f(x|\theta)$, the statistic $S(x)$ of x "is sufficient for θ if the conditional distribution of x given

$S(x) = s$ is independent of θ" (Scott and Nowak, 2004). This can be interpreted according to the following statement: "any inference strategy based on $f_\theta(x)$ may be replaced by a strategy based on $f_\theta(s)$" (Scott and Nowak, 2004).

Therefore, considering $\{x_1^n, Z_1^n\}$ instead of x, given the vector of sufficient statistics $s(x, Z)$, the EM algorithm for the m^{th} iteration becomes:

- **E-step:** set $\tilde{s}^{(m+1)} = E_{\tilde{P}}[s(x, Z)]$, where $\tilde{P}(Z) = P(Z|x, \theta^{(m)})$; (4.1)
- **M-step:** set $\theta^{(m+1)}$ to the θ with maximum likelihood given $\tilde{s}^{(m+1)}$.

These authors introduced an algorithm called *incremental EM* that was shown to reduce the number of iterations needed to converge. After having divided the original dataset $\Omega = \{x, Z\}$ in blocks $\{\Omega_1, ..., \Omega_n\} = \{x_1, Z_1, ..., x_n, Z_n\}$, the vector of sufficient statistics corresponding to each block Ω_j is initialized to an initial guess $s_j^{(0)}$. Thus, for the m^{th} iteration, the successive E and M steps are applied as follows:

- **E-step:** Select a block Ω_i to be updated;

  Set $\tilde{s}_j^{(m+1)} = \tilde{s}_j^{(m)}$ for every $j \neq i$; (4.2)

  Set $\tilde{s}_i^{(m+1)} = E_{\tilde{P}_i}[s(x, Z)]$, where $\tilde{P}_i(Z) = P(Z_i|x_j, \theta^{(m)})$; (4.3)

  Set $\tilde{s}^{(m+1)} = \tilde{s}_i^{(m+1)} + \tilde{s}_j$; (4.4)

- **M-step:** set $\theta^{(m+1)}$ to the θ with maximum likelihood given $\tilde{s}^{(m+1)}$;
- If convergence is reached, store $\tilde{s}_i = \tilde{s}_i^{(m+1)}$.

4.1.3 Incremental EM for SCFGs

In our context, the observed variables $\{x_i\}$ correspond to a given sequence, while the unobserved ones $\{Z_i\}$ correspond to the associated derivation trees $\{d_{x_i}\}$. Therefore,

in gEM, η (see Eq. 3.28) is a vector of sufficient statistics of θ, and $\tilde{s}$ can be identified with η. The E-step now consists in computing the vector η, while the M-step consists in reestimating the associated vector of production probabilities θ. It will be shown here how gEM can be modified to re-estimate these production probabilities incrementally.

For the m^{th} iteration, after having divided the original dataset Ω of sequences in blocks $\{\Omega_1, ..., \Omega_n\}$, each block containing a certain number of sequences, this concept may be adapted to gEM as follows:

- **E-step:** Select a block Ω_i to be updated;

  Compute $\eta^{(m+1)}$ using Alg. 18 and 19 on Ω_i (see Section 3.2.2);

  Set $\eta'^{(m+1)} = \eta^{(m+1)} + \sum_{j \neq i} \eta_j$; (4.5)

- **M-step:** re-estimate $\theta^{(m+1)}$ using Eq. 3.27 on $\eta'^{(m+1)}$;

- If convergence is reached, store $\eta_i = \eta^{(m+1)}$.

4.1.4 Incremental gEM (igEM)

Neal and Hinton's incremental algorithm updates the parameters by presenting the data sequentially, or by presenting the data in order to give preference to a block for which the algorithm has not yet stabilized. However this algorithm is not exactly incremental as defined in Chapter 2. Indeed, at every iteration, all the data is considered to compute $\tilde{s}^{(m+1)}$, while the purpose of this work is to learn new data not present at the beginning of the training.

The following approximation of Neal's algorithm is proposed. Consider that the SCFG probabilities have previously been learned from a block of sequences Ω_1, and that the final value of η, referred to as η_1, is stored. Then, to learn a new block Ω_2, the m^{th} iteration of the E and M-steps of the incremental EM algorithm, become:

- **E-step:** Compute $\eta^{(m+1)}$ using Alg. 18 and 19 on Ω_2;

$$\text{Set } \eta'^{(m+1)} = \eta^{(m+1)} + \eta_1; \tag{4.6}$$

- **M-step:** re-estimate θ using Eq. 3.27;

- If convergence is reached, store the new $\eta_2 = \eta'^{(m+1)}$.

The only difference with Neal's algorithm lays in the fact that Ω_2 was not considered during the initial estimation of θ. If another dataset Ω_3 is available after training has been led on $\{\Omega_1, ..., \Omega_2\}$, the procedure remains the same, using η_2 and applying the E-step on Ω_3.

A version of Alg. 17 that allows for incremental learning is given in Alg. 26. Note that in contrast to the original gEM, `Get-Inside-Probs()` is performed only on new data Ω_{i+1}, and `Get-Expectation()` produces η^{m+1}. The framed numbers highlight the difference with Alg. 17.

Algorithm 26: `Incremental gEM()`

1- load η_i;
2- `Get-Inside-Probs()` on Ω_{i+1};
3- $loglikelihood(0) = \sum_{x\in\Omega} \alpha_x(0, L(x)|Start)$;
4- **while** $loglikelihood^{(m)} - loglikelihood^{(m-1)} > \varepsilon$ **do**
  5- `Get-Expectation()`;
  6- $\eta'^{(m+1)} = \eta_i + \eta^{(m+1)}$;
  7- **foreach** $(A \rightarrow \lambda)$ **do**
    8- $\theta'(A \rightarrow \lambda) = \frac{\eta'^{(m+1)}(A\rightarrow\lambda)}{\sum_\mu \eta'^{(m+1)}(A\rightarrow\lambda')}$;
  9- $m = m + 1$;
  10- `Get-Inside-Probs()`;
  11- $loglikelihood(m) = \sum_{x\in\Omega} \alpha_x(0, L(x)|Start)$;
12- save $\eta_{i+1} = \eta^{(m)}$;

Suppose for example that training was completed successfully for the Mercury MFR (Annex 3.1) on a training block Ω_1 of 20 sequences, and that the final values $\eta_1(Na \rightarrow$

$S1\ T6) = 132.69$ and $\eta_1(Na \rightarrow Q6\ Q6) = 197.57$ are stored. Suppose that an iteration on a new block Ω_2 of 20 sequences gives $\eta^{(1)}(Na \rightarrow S1\ T6) = 83.38$ and $\eta^{(1)}(Na \rightarrow Q6\ Q6) = 148.49$. Thus, $\theta'(Na \rightarrow S1\ T6)$ is computed as follows:

$$\begin{aligned} \theta'(Na \rightarrow S1\,T6) &= \frac{\eta'^{(1)}(Na \rightarrow S1\,T6)}{\eta'^{(1)}(Na)} = \frac{\eta_1(Na \rightarrow S1\,T6) + \eta^{(1)}(Na \rightarrow S1\,T6)}{\eta_1(Na) + \eta^{(1)}(Na)} \\ &= \frac{132.69 + 83.38}{(132.69 + 197.57) + (83.38 + 148.49)} = 0.384 \end{aligned} \tag{4.7}$$

Suppose then that the next iteration on Ω_2 gives $\eta^{(2)}(Na \rightarrow S1\ T6) = 155.01$ and $\eta^{(2)}(Na \rightarrow Q6\ Q6) = 163.09$. Thus, $\theta'(Na \rightarrow S1\ T6)$ is now computed as follows:

$$\begin{aligned} \theta'(Na \rightarrow S1\,T6) &= \frac{\eta'^{(2)}(Na \rightarrow S1\,T6)}{\eta'^{(2)}(Na)} = \frac{\eta_1(Na \rightarrow S1\,T6) + \eta^{(2)}(Na \rightarrow S1\,T6)}{\eta_1(Na) + \eta^{(2)}(Na)} \\ &= \frac{132.69 + 155.01}{(132.69 + 197.57) + (155.01 + 163.09)} = 0.444 \end{aligned} \tag{4.8}$$

This process is repeated until convergence, and the final value $\eta_2(Na \rightarrow S1\ T6) = \eta'^{(convergence)}(Na \rightarrow S1\ T6) = \eta_1(Na \rightarrow S1\ T6) + \eta^{(convergence)}(Na \rightarrow S1\ T6)$ is stored for the next block of data.

4.2 On-line Incremental gEM

4.2.1 EM for Gaussian networks

Sato and Ishii (2000) proposes an on-line version of the EM algorithm for normalized gaussian networks. For this application, the batch EM algorithm can be adapted in the following way: while the E-step remains the same as for the general EM, the M-step

computes the weighed mean, with respect to the probability computed during the E-step, of functions that allow re-estimating the parameter of the network.

Imagine that we deal with an application in which parameters can be re-estimated using the weighted mean of a given function $f()$, that depends on both observable and hidden variables. The batch EM algorithm for such an application is the following:

- **The *Expectation* step (E-step):** given a set of observed variables $\{x_1, ..., x_n\}$ and a set of unobserved ones $\{Z_1, ..., Z_n\}$, it computes $P(Z_i|x_i, \theta)$, where θ is the parameter to estimate;
- **M-step:** it computes the weighted mean over n examples of a function of parameters x with respect to the posterior probability:

$$f^*(x, Z) = \frac{1}{n} \sum_{i=1}^{n} f(x_i) P(Z_i|x_i, \theta) \tag{4.9}$$

and it re-estimates the parameters using $f^*(x, Z)$.

4.2.2 On-line EM

The principle of the Sato's on-line algorithm consists in computing iteratively an estimate of $f^*(x, Z)$, that will be denoted herein $\tilde{f}(x, Z)$. The estimate $\tilde{f}(x, Z)$ can be derived from $f(x, Z)$ as shown in Eq. 4.10.

$$\begin{aligned} \tilde{f}(x, Z) &= \chi \sum_{i=1}^{n} \left(\prod_{j=i+1}^{n} \xi(j) \right) f(x_i, Z_i) P(Z_i|x_i, \theta(i-1)) \\ where \ \chi &= \left(\sum_{i=1}^{n} \prod_{j=i+1}^{n} \xi(j) \right)^{-1} \end{aligned} \tag{4.10}$$

$0 \leq \xi(j) \leq 1$ is a time dependent discount factor, χ is a normalization coefficient that plays the role of a learning rate, and $\theta(i-1)$ is the estimator of the parameter after the $i-1^{th}$ observation x_{i-1}.

This modified weighted mean can be computed in an iterative way, by presenting the examples x_i one after the other, using the Eq. 4.11. Consider that $\tilde{f}(x_1^i, Z_1^i)$ has already been computed on $\{x_1, ..., x_i, Z_1, ..., Z_i\}$:

$$\tilde{f}(x_1^{i+1}, Z_1^{i+1}) = \tilde{f}(x_1^i, Z_1^i) + \chi(i+1)\left(f(x_{i+1}, Z_{i+1})P(Z_{i+1}|x_{i+1}, \theta(i)) - \tilde{f}(x_1^i, Z_1^i)\right) \tag{4.11}$$

It has been shown that, with an appropriate value of χ, the modified weighted mean is equal to the classical mean, and that this on-line EM algorithm is equivalent to the batch version. It has also been proved that this on-line algorithm converges to the maximum likelihood estimator.

Considering a dataset $\Omega = x_1, ..., x_n$,and the fact that training has already been completed on $\{x_1, ...x_i\}$ the on-line EM algorithms can be applied as follows:

- **E-step:** compute $P(Z_{i+1} = j|x_{i+1}, \theta(i))$;
- **M-step:** compute $\tilde{f}(x_1^{i+1}, Z_1^{i+1})$ using Eq. 4.11 and re-estimate the parameters.

4.2.3 On-line gEM for SCFGs

The on-line EM method can be applied to gEM, where x_i is a given sequence, and where Z_i corresponds to d_{x_i} in the following way. The weighting probability $P(d_x|x, \theta)$ then corresponds to the probability of having a particular derivation tree d_x, given x and θ, and

f is now identified with the frequency of the rules, N. Indeed, Bayes' theorem gives the relation between N^* and η for a particular production rule $A \rightarrow \lambda$ and a set of derivation trees $d_\Omega = \{d_x\}$:

$$\begin{aligned} N^*(A \rightarrow \lambda, d_\Omega) &= \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) P(d_x | x, G_s) \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x | G_s)} \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x, \Delta_x | G_s)} \\ &= \frac{\eta(A \rightarrow \lambda, d_\Omega)}{|\Omega|} \end{aligned} \tag{4.12}$$

In the this application, however, all the derivation trees have to be considered to re-estimate the production rule probabilities. Indeed, for each sequence, we have to sum the weighted frequency of the rules over all the corresponding derivation trees, as follows:

$$\begin{aligned} N^*(A \rightarrow \lambda) &= \sum_{d_\Omega \in \Delta_\Omega} N^*(A \rightarrow \lambda, d_\Omega) \\ &= \sum_{d_\Omega \in \Delta_\Omega} \frac{1}{|\Omega|} \sum_{x \in \Omega} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x, \Delta_x | G_s)} \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) \frac{P(x, d_x | G_s)}{P(x, \Delta_x | G_s)} \\ &= \frac{1}{|\Omega|} \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s) \\ &= \frac{\eta(A \rightarrow \lambda)}{|\Omega|} \end{aligned} \tag{4.13}$$

Thus, the re-estimation equation (Eq. 3.27) can be modified according to Eq. 4.13:

$$\theta(A \rightarrow \lambda) = \frac{\eta(A \rightarrow \lambda)}{\eta(A)} = \frac{\eta(A \rightarrow \lambda)/|\Omega|}{\eta(A)/|\Omega|} = \frac{N^*(A \rightarrow \lambda)}{N^*(A)}$$

$$\text{where} \qquad N^*(A) = \sum_{\lambda} N^*(A \rightarrow \lambda) \tag{4.14}$$

Suppose that $\tilde{N}_i(A \rightarrow \lambda)$ (obtained from $\{x_1, ..., x_i\}$) has already been computed. Using Eq. 4.13, it is now possible to compute $\tilde{N}_{i+1}(A \rightarrow \lambda)$ iteratively, and the modified version of $N^*(A \rightarrow \lambda)$ is computed on $\{x_1, ..., x_i\}$, as follows:

$$\begin{aligned}
\tilde{N}_{i+1}(A \rightarrow \lambda) &= \tilde{N}_i(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\sum_{d_{x_{i+1}} \in \Delta_{x_{i+1}}} N(A \rightarrow \lambda, d_{x_{i+1}}) P(d_{x_{i+1}} | x_{i+1}, G_s) \\
&\quad -\tilde{N}_i(A \rightarrow \lambda)] \\
&= \tilde{N}_i(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\eta(A \rightarrow \lambda, x_{i+1}) - \tilde{N}_i(A \rightarrow \lambda)]
\end{aligned} \tag{4.15}$$

Based on Eq. 4.15, the on-line EM algorithm can be adapted to gEM in the following way. Assuming that training has already been completed on $\{x_1, ...x_i\}$, the m^{th} iteration gives:

- **E-step:** Compute η_{i+1} using Alg. 18 and 19 on x_{i+1};
  Set $\tilde{N}_{i+1} = \tilde{N}_i + \chi(i+1)[\eta_{i+1} - \tilde{N}_i]$; (4.16)

- **M-step:** re-estimate $\theta^{(i+1)}(A \rightarrow \lambda) = \frac{\tilde{N}_{i+1}(A \rightarrow \lambda)}{\tilde{N}_{i+1}(A)}$; (4.17)

- If convergence is reached, store $\tilde{N}_{i+1}$.

4.2.4 On-line incremental gEM (oigEM)

The algorithm described until now is said to be on-line algorithm because it consists in presenting the examples of the training dataset one after the other, from the first to the last one, and looping until convergence. However, once more, it is not incremental as defined in Sec. 2. First, this on-line algorithm can be modified in order to make it sequential for blocks of data instead of only single examples. Consider two data blocks $\{\Omega_1, ..., \Omega_i\}$ and Ω_{i+1} of sizes $|\Omega_{1,...,i}|$ and $|\Omega_{i+1}|$ and suppose that $\tilde{N}_i(A \rightarrow \lambda)$, after training on $\{\Omega_1, ..., \Omega_i\}$, was already computed.

$$\begin{aligned}
\tilde{N}_{i+1}(A \rightarrow \lambda) &= \tilde{N}_i(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\frac{\sum_{x \in \Omega_{i+1}} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(d_x|x, G_s)}{|\Omega_{i+1}|} \\
&\quad -\tilde{N}_i(A \rightarrow \lambda)] \\
&= \tilde{N}_{i+1}(A \rightarrow \lambda) \\
&\quad +\chi(i+1)[\frac{\sum_{x \in \Omega_{i+1}} \eta(A \rightarrow \lambda, x)}{|\Omega_{i+1}|} - \tilde{N}_i(A \rightarrow \lambda)] \\
&= \tilde{N}_{i+1}(A \rightarrow \lambda) + \chi(i+1)[\frac{\eta_{i+1}(A \rightarrow \lambda)}{|\Omega_{i+1}|} - \tilde{N}_i(A \rightarrow \lambda)] \quad (4.18)
\end{aligned}$$

Thus, inspiring from the incremental EM (Neal and Hinton, 1998) and taking in account the fact that some scheme for which the algorithm has not yet stabilized can be privileged, the following incremental algorithm can be defined. For each block, the SCFG is trained over several iterations until convergence. Suppose that training has already been successfully performed on the first block Ω_1, and that, for each rule $A \rightarrow \lambda$, $\tilde{N}_1(A \rightarrow \lambda)$ – the final value of $\tilde{N}(A \rightarrow \lambda)$ – is stored. In order to learn a new block Ω_2, $\tilde{N}_2(A \rightarrow \lambda)$ can be computed for the m^{th} iteration according to:

- **E-step:** Compute $\eta^{(m+1)}$ using Alg. 18 and 19 on Ω_2;

$$\text{Set } \tilde{N}^{(m+1)} = \tilde{N}_1 + \chi(m+1)[\frac{\eta_2^{(m+1)}}{|\Omega_2|} - \tilde{N}_1]; \tag{4.19}$$

- **M-step:** re-estimate $\theta'(A \rightarrow \lambda) = \dfrac{\tilde{N}^{(m+1)}(A \rightarrow \lambda)}{\tilde{N}^{(m+1)}(A)}$; (4.20)

- If convergence is reached, store $\tilde{N}_2 = \tilde{N}^{(m+1)}$.

A version of Alg. 17 that allows for on-line incremental learning is given in Alg. 27. In this algorithm, steps allow to manage updates to the value $\tilde{N}$. Note that in contrast to the original gEM, `Get-Inside-Probs()` is performed only on new data Ω_{i+1}, and `Get-Expectation()` produces $\eta^{(m+1)}$. The framed numbers highlight the difference with Alg. 17.

Algorithm 27: `On-line incremental gEM()`

1- load $\tilde{N}_i$;
2- `Get-Inside-Probs()` on Ω_{i+1};
3- $loglikelihood(0) = \sum_{x \in \Omega} \alpha_x(0, L(x)|Start)$;
4- **while** $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  5- `Get-Expectation()` ;
  6- **foreach** $(A \rightarrow \lambda)$ **do**
    7- $\tilde{N}^{(m+1)}(A \rightarrow \lambda) = \tilde{N}_i(A \rightarrow \lambda) + \chi(m+1)[\frac{\eta^{(m+1)}(A \rightarrow \lambda)}{|\Omega_{i+1}|} - \tilde{N}_i(A \rightarrow \lambda)]$;
    8- $\theta(A \rightarrow \lambda) = \frac{\tilde{N}^{(m+1)}(A \rightarrow \lambda)}{\tilde{N}^{(m+1)}(A)}$;
  9- $m = m + 1$;
  10- `Get-Inside-Probs()`;
  11- $loglikelihood(m) = \sum_{x \in \Omega} \alpha_x(0, L(x)|Start)$;
13- save $\tilde{N}_{i+1}$;

Suppose that training was completed successfully for the Mercury MFR (Annex 3.1) on a training block Ω_1 of 20 sequences, and that the final values $\tilde{N}_1(Na \rightarrow S1\ T6) = \eta_1(Na \rightarrow S1\ T6)/20 = 132.69/20 = 6.63$ and $\tilde{N}_1(Na \rightarrow Q6\ T6) = \eta_1(Na \rightarrow$

$Q6\ Q6)/20 = 197.57/20 = 9.88$ are stored. Suppose that an iteration on a new block Ω_2 of 20 sequences gives $\eta^{(1)}(Na \rightarrow S1\ T6)/20 = 83.38$ and $\eta^{(1)}(Na \rightarrow Q6\ Q6)/20 = 148.49$. Thus, for $\chi = 0.25$, $\tilde{N}^{(1)}(Na \rightarrow S1\ T6)$ and $\tilde{N}^{(1)}(Na \rightarrow Q6\ T6)$ are computed as follows:

$$\begin{aligned}
\tilde{N}^{(1)}(Na \rightarrow S1\ T6) &= \tilde{N}_1(Na \rightarrow S1\ T6) \\
&\quad +0.25 \cdot (\eta^{(1)}(Na \rightarrow S1\ T6)/20 - \tilde{N}_1(Na \rightarrow S1\ T6)) \\
&= 6.63 + 0.25 \cdot (83.38/20 - 6.63) = 6.01 \\
\tilde{N}^{(1)}(Na \rightarrow Q6\ T6) &= \tilde{N}_1(Na \rightarrow Q6\ Q6) \\
&\quad +0.25 \cdot (\eta^{(1)}(Na \rightarrow Q6\ Q6)/20 - \tilde{N}_1(Na \rightarrow Q6\ Q6)) \\
&= 9.88 + 0.25 \cdot (148.49/20 - 9.88) = 9.27
\end{aligned} \tag{4.21}$$

$\theta'(Na \rightarrow S1\ T6)$ is computed as follows:

$$\theta'(Na \rightarrow S1\ T6) = \frac{\tilde{N}^{(1)}(Na \rightarrow S1\ T6)}{\tilde{N}^{(1)}(Na)} = \frac{6.01}{6.01 + 9.27} = 0.393 \tag{4.22}$$

Suppose then that the next iteration on Ω_2 gives $\eta^{(2)}(Na \rightarrow S1\ T6) = 161.51$ and $\eta^{(2)}(Na \rightarrow Q6\ Q6) = 156.99$. Thus, for $\chi = 0.25$, $\tilde{N}^{(1)}(Na \rightarrow S1\ T6)$ and $\tilde{N}^{(1)}(Na \rightarrow Q6\ T6)$ are computed as follows:

$$\begin{aligned}
\tilde{N}^{(1)}(Na \rightarrow S1\ T6) &= \tilde{N}_1(Na \rightarrow S1\ T6) \\
&\quad +0.25 \cdot (\eta^{(2)}(Na \rightarrow S1\ T6)/20 - \tilde{N}_1(Na \rightarrow S1\ T6)) \\
&= 6.63 + 0.25 \cdot (161.51/20 - 6.63) = 6.99
\end{aligned}$$

$$\begin{aligned}\tilde{N}^{(1)}(Na \rightarrow Q6\,T6) &= \tilde{N}_1(Na \rightarrow Q6\,Q6) \\ &\quad +0.25 \cdot (\eta^{(2)}(Na \rightarrow Q6\,Q6)/20 - \tilde{N}_1(Na \rightarrow Q6\,Q6)) \\ &= 9.88 + 0.25 \cdot (156.99/20 - 9.88) = 9.37 \end{aligned} \quad (4.23)$$

$\theta'(Na \rightarrow S1\,T6)$ is computed as follows:

$$\theta'(Na \rightarrow S1\,T6) = \frac{\tilde{N}^{(2)}(Na \rightarrow S1\,T6)}{\tilde{N}^{(2)}(Na)} = \frac{6.99}{6.99 + 9.37} = 0.427 \quad (4.24)$$

This process is repeated until convergence, and the final value $\tilde{N}_2(Na \rightarrow S1\,T6) = \tilde{N}^{(convergence)}(Na \rightarrow S1\,T6)$ is stored for the next block of data.

4.3 Comparison of igEM and oigEM

The main difference between igEM and oigEM lies in the fact that parameter χ must be set in the second one. Parameter χ allows giving more importance either to the new dataset, or the old one, and thereby tuning the algorithm. A brief calculation easily show that setting $\chi(i) = \frac{\chi(i-1)}{1+\chi(i-1)}$ for the i^{th} block of sequences, make the two algorithms identical. In the original on-line EM algorithm, in which the examples are presented one after the other, χ is supposed to decrease, in order to give less importance to the new data. Indeed, if learning has already been performed on a lot of examples, the new one should not have as much importance as the old data. However, in radar ES applications, radar sequences can be obtained in different conditions and environment, and one could therefore have more or less confidence in a new block of sequences. That is why, if one has very good confidence in a new block, he could decide to increase χ.

The main advantage of these two algorithms lies in the fact that local optima may be avoided thanks to the re-initialization of the production probabilities of the SCFG when

new data is added. Suppose that training was previously performed on an original dataset Ω_1 and that a new one, Ω_2 becomes available. Both igEM and oigEM use only the support graphs from Ω_2 to compute η_2, while information on Ω_1 is already stored in η_1. Consider that each dataset is not fully representative of the whole problem, and that their distributions can even be disjoined in the space of features. In this case, as illustrated in Fig 14, the production rule probabilities obtained after training on Ω_1 should not serve as the starting point for the incremental learning process on Ω_2. In this case, it can lead to being trapped in a local minimum of the new cost function associated with $\Omega_1 + \Omega_2$, unless probabilities are re-initialized. Suppose that the plain curve represents the cost function associated with a system trained on block Ω_1, and that the dotted curved represents the cost function associated with a system first trained on block Ω_1 and then incrementally on a new block Ω_2. Point (1) represents the idéal solution of an optimization performed on Ω_1. It appears clearly that if this point is used as a starting point for training on Ω_2 (point (2)), then it will lead to the local optimum at point (3). On the other hand, if the probabilities are randomly re-initialized before training on Ω_2, allowing, for instance, to start at point (4), the optimization would lead to point (5), and a local optimum would be avoided. Other approaches, such as the gradient descent technique of Baldi and Chauvin (1994), do not re-initialize the probabilities prior to learning new training sequences, and were therefore not considered.

The HOLA algorithm uses Ω_2 to update a histogram created using Ω_1. In this case, only one distribution is used during the whole process. Note that this distribution is the only one to be modified when new data is learned, and that it has a fixed size, making HOLA very efficient for incremental learning.

An incremental version of gEM(VS) is straightforward – gEM(VS) would require deriving new routines `Get-Inside-Probs` and `Get-Expectation` (`Get-Inside-Probs-VS` and `Get-Expectation-VS`) in order to compute $\hat{\eta}$ for each sequence of the dataset,. The value $\hat{\eta}$ is the value η corresponding to the

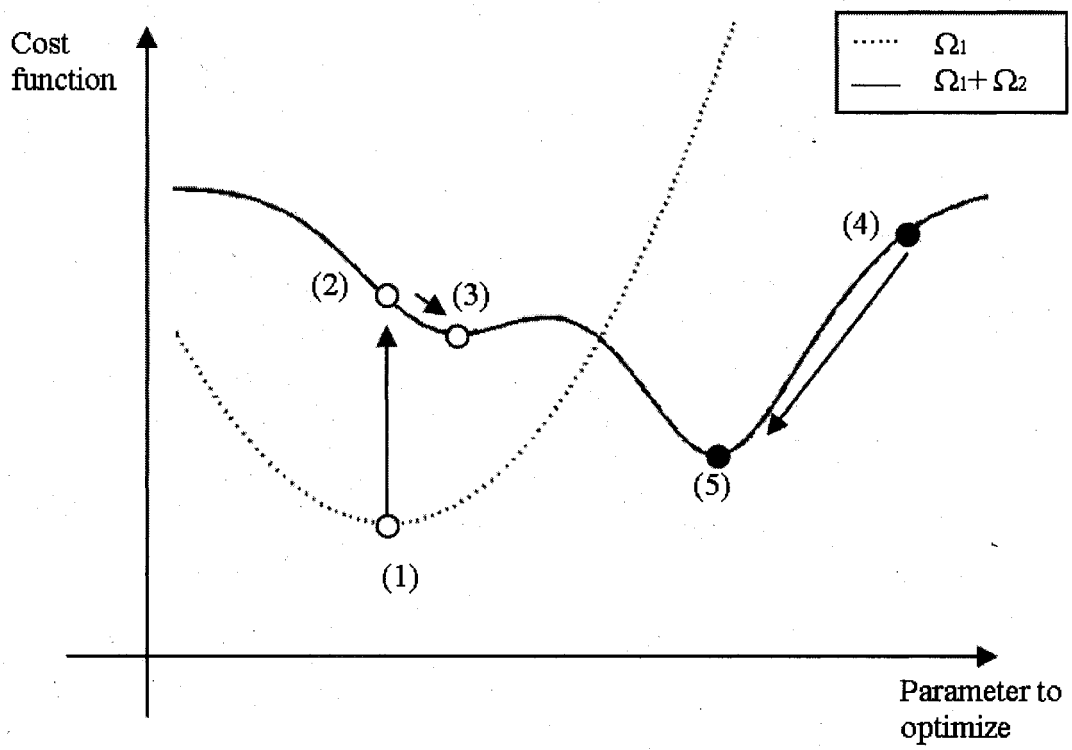


Figure 14 Illustration of the importance of re-initializing probabilities when training on a new block of sequences.

derivation tree of maximum likelihood. Therefore, adapting igEM and oigEM to VS would only involve $\hat{\eta}$ instead of η in Eqs. 3.27 and 4.20 respectively. Finally, one can see that igEM and oigEM can directly be adapted to TS. Indeed, TS computes the different elements of Eq. 3.2, which are also the elements of η.

Table 4.3 displays a summary of the re-estimation formulas of the incremental learning techniques, discussed in this chapter, and their relation with the global re-estimation formula of Eq.3.2. This table also contains definitions of the symbols used in each reestimation formula. The relationship between igEM and oigEM appears clearly in this table.

Table II

Summary of re-estimation formulas of techniques for incremental learning of SCFGs.

Method	Reestimation Formula
General	$\theta'(A \rightarrow \lambda) = \frac{\sum_{x \in \Omega} \frac{\sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x \mid G_s)}{P(x, \Delta_x \mid G_s)}}{\sum_{x \in \Omega} \frac{\sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x \mid G_s)}{P(x, \Delta_x \mid G_s)}}$ $N(A \rightarrow \lambda, d_x)$ = frequency of the rule $A \rightarrow \lambda$ in the derivation tree d_x
igEM	$\theta'(A \rightarrow \lambda) = \frac{\eta_0(A \rightarrow \lambda) + \eta_1(A \rightarrow \lambda)}{\sum_\lambda \eta_0(A \rightarrow \lambda) + \eta_1(A \rightarrow \lambda)}$ $\eta_0(A \rightarrow \lambda) = \eta(A \rightarrow \lambda)$ computed on the previous dataset and stored $\eta_1(A \rightarrow \lambda) = \eta(A \rightarrow \lambda)$ computed on the new dataset and updated
oigEM	$\theta'(A \rightarrow \lambda) = \frac{\tilde{N}(A \rightarrow \lambda)}{\tilde{N}(A)}$ $\tilde{N}(A \rightarrow \lambda) = \tilde{N}_0(A \rightarrow \lambda) + \chi[\frac{\eta_1(A \rightarrow \lambda)}{\lvert\Omega_i\rvert} - \tilde{N}_0(A \rightarrow \lambda)]$ $\tilde{N}(A \rightarrow \lambda)$ = balanced frequency of the rule $A \rightarrow \lambda$
HOLA	$\theta'(A \rightarrow \lambda) = \theta(A \rightarrow \lambda) \cdot (1 + \chi \cdot (N(A \rightarrow \lambda).o - N(A \rightarrow \lambda).s)$ $N(A \rightarrow \lambda).o$ = frequency of the rule $A \rightarrow \lambda$ over the training set $N(A \rightarrow \lambda).s$ = frequency of the rule $A \rightarrow \lambda$ over the generated set

CHAPTER 5

EXPERIMENTAL METHODOLOGY

In order to characterize and compare the performance of different techniques presented in this thesis for learning production rule probabilities, an experimental framework has been developed. First, this chapter describes the synthetic MFR data used for proof-of-concept computer simulations. Then, two experimental protocols used during computer simulations is presented. The first one is used for batch learning simulations (for the first part of this thesis), whereas the second one is used for incremental learning simulations (for the second part of this thesis). Finally the measures used to assess the performance of learning techniques are detailed.

5.1 MFR radar data

Through this thesis, the hypothetical radar ES system shown in Fig. 4 was assumed for computer simulations. It consists of a TOA receiver, a tokenizer, and a sequence recognition module. The purpose of this work is to train word level SCFGs for the sequence recognition module. It was assumed that sequences of pulses were already received by the TOA receiver, and processed by a tokenizer to identify the corresponding words. In real radar ES applications, the data would come from the battlefield, and would therefore contain imperfections, which would result in imperfect sequences of words. It was assumed herein that an ES analyst would observe the data in order to eliminate any of these imperfection, therefore providing the sequences of words actually emitted by the radar.

Data for computer simulations was generated using the Automatic Radar Emitter Recognition (ARER) software prototype v02 provided by DRDC-Ottawa. The ARER software was developed by the Adaptive Systems Laboratory of McMaster University and can generates the TOA for sequences of synthetic radar pulses according to several MFRs.

Data generated with this software emulates data sources consisting of pulses or words from different MFRs. These two MFRs are fictitious but representative of MFR families. At a pulse level, Mercury has a fixed pattern structure, with fixed words durations, while Pluto represents a family of MFR with variable pattern, with variable words durations. At a word level, this means that Pluto's words are harder to identify and that a detected sequence of words from Pluto would have more noise in general than a detected sequence of words from Mercury.

This program was originally designed to account for the fact that an MFR can detect several targets at the same time using different tracks. To simplify the problem, it was decided to concentrate on only one target. The fact that the state of a phrase from a given track can influence phrases from other tracks was not considered. The duration of a state is set using gaussian distributions with specific means and variances, that approximate reality. The rest of this section presents the three MFRs used for simulations. They are named Mercury, Pluto and Venus.

In this section, the three different MFRs used for the experiments, namely Mercury, Pluto, and Venus – that is derived in two versions, named VenusA and VenusB –, are successively described from Section 5.1.1 to 5.1.3. Finally, the definition of the ambiguity of the grammars is given in Section 5.1.5.

5.1.1 The Mercury data

The Mercury MFR has a vocabulary of nine distinct words. Each word consists of a characteristic pattern of pulses and all nine words have a common, fixed temporal duration of 75000 crystal counts, as shown in Fig. 15). Parts A, C and E are dead time in which no pulse is emitted. Part B is a set of pulse emitted with a fixed pulse repetition interval (PRI) for each word. It can comprise between 65 and 675 pulses according to the word. Part D is a burst of 12 pulses emitted with a fixed PRI that is common to all nine words.

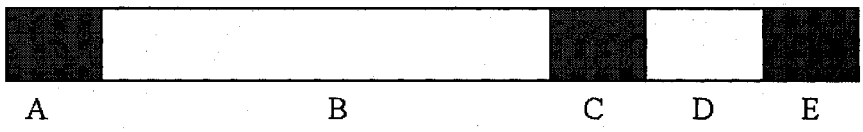


Figure 15 Pulse envelopes of Mercury words

The Mercury MFR can be in one of five functional states – `Search (S)`, `Acquisition (Acq)`, `Non-Adaptive Track (Na)`, `Range Resolution (Rr)`, and `Track Maintenance (Tm)`. Each state emits a phrase of four words. If the radar is in `S`, it can remain there, or move to `Acq` once a target is detected. The target acquisition cycle involves transitions from `Acq` to `Na` to `Rr` and finally to `Tm`. The radar can remain in any of these states for an unspecified length of time. Finally, the target acquisition or track can be abandoned at any point, at which time the radar returns to `S`. The structures of the phrases associated to the states are briefly given by Haykin and Currie (2003):

- `Search (S)`: the words are cycled through the quadruplet W1-W2-W4-W5 or the triplet W1-W3-W5, which can begin by any of the four or three words.
- `Acquisition (Acq)`: all the words are converted to the same variety of words, one of W1, W2, W3, W4, W5 or W6.
- `Non-adaptive track (Na)`: all the words are converted to W6.
- `Range resolution (Rr)`: the first word will hop between W7, W8 and W9, while the last three words will stay on W6.
- `Track maintenance (Tm)`: all the words are converted to W6, W7, W8 or W9.

A word-level CFG was designed for this MFR from its functional description (Haykin and Currie, 2003) (see Annex 3.1), according to the Chomsky Normal Form (as required by all

techniques of interest). The structure of the grammar is important. It is designed such that the states appear as non-terminals, and that they are easy to find in a given derivation tree, in order to allow for detection of threat. For reference, the Mercury language is presented in Annex 3.1.2.

To apply learning techniques, the empty string ϵ was considered as a nonterminal and a tenth word was added to each one of the original sequences. The following rule was then added:

$$\epsilon \rightarrow 10$$

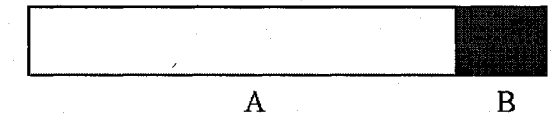
A dataset was generated for Mercury using the ARER software. It consists of 400 sequences, where each sequence corresponds to the set of words that would be produced by this MFR during one target detection, while switching through all its internal states, starting and ending in `S`. Mercury sequences have a length that ranges from 108 to 1540 words, with an average of 588 words.

This data set was then partitioned into four equal parts – a training subset `MTrain`, a validation subset `MVal`, a test subset `MTest`, and an *ideal* test subset `MTI`. Inside an ES system, the MFRs words would first be detected from within the stream of intercepted radar pulses, and then sequences of words would be recognized using the SCFGs. Prior to sequence recognition, errors occur if (1) a word is incorrectly detected, (2) a word is missing or not detected, or (3) multiple words are detected simultaneously. If only the best-matching detection is retained for sequence recognition, these 3 cases are equivalent to incorrectly detected words (case 1). Accordingly, two noisy versions of `MTI` – $\texttt{MTN}_1$ and $\texttt{MTN}_2$ – were generated. Each noisy test set consists of the 100 sequences from `MTI`. Then, to select incorrectly detected words to be modified, a Poisson process was applied with a mean of 50 words for $\texttt{MTN}_1$ and 10 words for $\texttt{MTN}_2$. Finally, a uniform law was used to

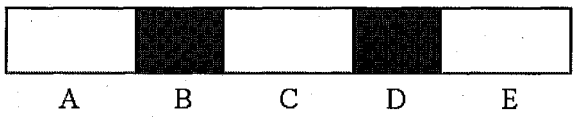
determine the replacement words. For each sequence in each database, the corresponding states of the MFR were stored for reference.

5.1.2 The Pluto data

The Pluto MFR has a vocabulary of 6 distinct words, composed of 4 active words, a blank word containing no pulse and a termination character. The pulse envelope of an active word is represented in Fig. 16(a). It is composed of two parts. Part A is a set of pulses whose number can vary between 333 and 415 according to the word. Part B is a dead time with a duration sufficient to complete the word – depending on the position of a word in a sequence, its total dwell time varies. The termination character is composed of 5 parts, as shown in Fig. 16(b). Parts B and D are dead time. Parts A, C and E are sets of respectively 5, 8 and 12 pulses, each one with a different fixed PRI.



(a) Pulse envelope of a Pluto word



(b) Pulse envelope of a Pluto termination character

Figure 16 Pulse envelopes of Pluto words and termination character.

The Pluto MFR can be in one of four functional states – `Search (S)`, `Non-Adaptive Track (Na)`, `Range Resolution (Rr)`, and `Track Maintenance (Tm)`. Each state emits a phrase of 3 words: a blank or active word of 51000 crystal counts, a second active word of 50000 crystal counts, and a termination character. If the radar is in `S`, it can remain there, or move to `Na` once a target is detected. Thus, the radar will hop from `Na` to `Rr` and finally to `Tm`. The radar can remain in any of these states for an unspecified length of time. Finally, the target track can be abandoned at any point, at

which time the radar returns to S. The structures of the phrases associated to these states are briefly given by Haykin and Currie (2003):

- Search (S): the words are cycled through the quadruplet W2-W2-W5.
- Non Adaptive Track (Na): the words are cycled through the quadruplet W2-W2-W5.
- Range resolution (Rr): the first word will change to W0, while the second word will hop between W1, W2, W3 and W4. The last word remains W5.
- Track maintenance (Tm) :the first word will change to W2, while the second word will hop between W1, W2, W3 and W4. The last word remains W5.

A word-level CFG was designed for this MFR from its functional description (Haykin and Currie, 2003) (see Annex 3.1.4), according to the Chomsky Normal Form. As for Mercury, a last seventh word was added to each sequence. For reference, the Pluto language is presented in Annex 3.1.5.

A data set was generated for Pluto using the ARER software. It consists of 400 sequences, where each sequence corresponds to the set of words that would be produced by this MFR during one target detection, while switching through all its internal states, starting and ending in S. Pluto sequences have a length that ranges from 397 to 953 words, with an average of 644 words.

As with the Mercury data, this dataset was then partitioned into four equal parts – a training subset PTrain, a validation subset PVal, a test subset PTest, and an *ideal* test subset PTI. As for Mercury, two noisy versions of PTI – PTN_1 and PTN_2 – were generated. Each noisy test set consists of the 100 sequences from PTI. Then, to select incorrectly detected words to be modified, a Poisson process was applied with a mean of 50 words for PTN_1 and 10 words for PTN_2. Finally, a uniform law was used to determine the replacement

words. For each sequence in each database, the corresponding states of the MFR were stored for reference.

5.1.3 The Venus data

The ARER software that was provided was only able to generate sequences corresponding to the Mercury and Pluto MFRs. Given the need for assessing the capacity of the trained SCFGs to recognize the associated MFR system among several other MFR systems, two other fictitious MFRs were simulated. The description of these MFR was inspired by the formal description of an MFR provided by DRDC-Ottawa, and are referred as VenusA and VenusB. Since these MFRs were not provided by DRDC-Ottawa, and only their word level functioning is of interest in this work, the structures of their words are not described.

VenusA can be in two different states, `General Search (GS)` and `Tm`, that cycle in the following order: `Gs` $\rightarrow$ `Tm` $\rightarrow$ `Gs`. VenusB can be in four different states, `General Search (GS)`, `Directed Search (DS)`, `Acq` (which is decomposed in two sub-states: $\texttt{Acq}_1$ and $\texttt{Acq}_2$) and `Tm`, that cycle in the following order: `Gs` $\rightarrow$ `Ds` $\rightarrow$ $\texttt{Acq}_1$ $\rightarrow$ $\texttt{Acq}_2$ $\rightarrow$ `Tm` $\rightarrow$ `Gs`.

VenusA and VenusB languages are described in Annexes 3.2 and 3.3. Since these MFRs are only used for tests, and not for training, there was no need for designing corresponding CFGs. In each case, a dataset of 100 sequences, named `VATI` for VenusA and `VBTI` for VenusB, were generated. The durations of the different states were set using a Poisson law, except for $\texttt{Acq}_2$ whose duration is fixed, using the following means:

- mean(`Gs`) = 130 phrases for VenusA / 90 phrases for VenusB;
- mean(`Ds`) = 70 phrases;
- mean(`Acq`) = 20 phrases;
- mean(`Tm`) = 100 phrases for VenusA / 80 phrases for VenusB.

For each state, the choice of the corresponding emitted phrases was determined using uniform laws over the possible words.

5.1.4 Languages conversion

In radar ES applications, the tokenizer shown in Fig. 4 would receive a sequence of pulses from a determined radar and gives the most probable sequences of words corresponding to each radar of its library. This can also be interpreted as follows: one original sequence is converted by the tokenizer into several sequences in the languages of the MFRs of its library. Suppose that a radar ES system's library only contains the descriptions of Mercury and Pluto. Therefore, an intercepted sequence of pulses would be translated into a sequence corresponding to the most probable sequence of words belonging to the Mercury language, and into another sequence corresponding to the most probable sequence of words belonging to the Pluto language.

In order to model this functioning, the sequences of `PTI`, `PTN1`, `PTN2`, `VATI` and `VBTI` were translated into Mercury language, and the sequences from `MTI`, `MTN1`, `MTN2`, `VATI` and `VBTI` were translated into Pluto language, using the conversion tables presented in Table III. This conversion process allows to create 6 databases of 400 sequences in the corresponding MFR language, belonging to the 4 different MFRs:

- `MROCTI` (containing the translation of `PTI`, `VATI` and `VBTI`);
- `MROCTN1` (containing the translation of `PTN1`, `VATI` and `VBTI`);
- `MROCTN2` (containing the translation of `PTN2`, `VATI` and `VBTI`);
- `PROCTI` (containing the translation of `MTI`, `VATI` and `VBTI`);
- `PROCTN1` (containing the translation of `MTN1`, `VATI` and `VBTI`);
- `PROCTN2` (containing the translation of `MTN2`, `VATI` and `VBTI`.

Table III

Conversion tables used to convert the MFRs vocabulary to those of Mercury and Pluto.

Mercury words	1	2	3	4	5	6	7	8	9
Corresponding Pluto words	4	3	4	5	6	1	2	3	6

Pluto words	0	1	2	3	4	5
Corresponding Mercury words	9	7 or 8	5 or 1	2 or 3	4	6

VenusA words	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Corresponding Mercury words	9	1	4	2	9	3	6	1	7	5	6	8	6	4
Corresponding Pluto words	6	4	1	6	5	4	2	3	3	4	1	2	4	2

VenusB words	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Corresponding Mercury words	2	1	6	9	4	3	6	7	3	7	5	8	4	5
Corresponding Pluto words	6	6	5	1	6	2	1	3	6	2	4	5	2	4

5.1.5 Ambiguity

A grammar is said to be ambiguous if several derivation trees can lead to the same sequence. In radar ES application, an MFR's grammar is ambiguous if two consecutive states can produce a same phrase. Each MFR's grammar was designed so that only one derivation tree rooted at a given radar state could produce a given phrase. This means that the grammars are unambiguous when considering the rules depending on a specified state. However, two different states can nevertheless lead to the same phrase. If these two states are consecutive, ambiguity appears – if the shared phrase appears two consecutive times, then the corresponding state transition cannot be uniquely determined. But if these two states are not consecutive, the intermediate states prevent ambiguity. Thus, the Pluto grammar is more ambiguous than Mercury grammar because it satisfies these conditions more often. A measure of the ambiguity can be defined as the number of phrases shared by two consecutive states divided by the total number of phrases emitted by these two states. Accordingly, Mercury has an ambiguity of 0.2 and Pluto has an ambiguity of 1. Even if no

grammar was designed for VenusA and VenusB, it can be seen in Tables XXII and XXIII (see Annex 3.2 and 3.3) that that their ambiguity would be null.

5.2 Experimental protocol for computer simulations

5.2.1 Protocol for batch learning

The experimental protocol used in the first part of this thesis involves batch learning of SCFG probabilities. During each trial, the prior probabilities of a SCFG are initialized, and training is performed over several iterations with `MTrain` (`PTrain`), using the hold out validation strategy. That is, until the difference between the negative log likelihoods of the sequences (for gEM and TS) or relative entropy of the distributions (for HOLA) obtained with `MVal` (`PVal`) is lower than 0.001 for two successive iterations. In order to assess the impact on performance of training set size, the number of training subset sequences from `MTrain` (`PTrain`) used during learning is progressively increased from 25 to 100 sequences, by increments of 25, while the sizes of corresponding validation and test subsets are held fixed. At the end of each trial, SCFG probabilities associated with the minima of the negative log likelihoods (gEM and TS), or of the relative entropy of words (HOLA) on `MTrain` (`PTrain`) are stored.

Each independent trial is replicated with production rule probabilities of a SCFG initialized in 10 different ways – one in a uniform way, and the remainder in a random way. `MTest` (`PTest`) is then used to assess performance and to select, for each technique, the "best" set of SCFG production rule probabilities (the set of probabilities that leads to the lowest perplexity – this apply even to HOLA) among the 10 different probability initializations. Average results, with corresponding standard error, are always obtained, as a result of the 10 independent simulation trials. Perplexity and recognition of radar states are assessed using `MTI`, $\texttt{MTN}_1$ and $\texttt{MTN}_2$ (`PTI`, $\texttt{PTN}_1$ and $\texttt{PTN}_2$), with the best set of SCFG probabilities. Finally, the capacity to recognize the MFR corresponding to the trained

grammar is assessed using MROCTI, MROCTN_1 and MROCTN_2 (PROCTI, PROCTN_1 and PROCTN_2).

The diagram of Fig. 17 summarizes the experimental protocol used to assess performance in the first part of this thesis.

5.2.2 Protocol for incremental learning

The experimental protocol used in the second part of this thesis involves incremental learning of SCFG probabilities. Subsets MTrain, MVal, PTrain, and PVal are subdivided in subdatasets of 5, 10, 20, 25 or 50 sequences. This gives new training subdatasets called:

- MTrain5(i), MVal5(i), PTrain5(i), and PVal5(i), for $i = 1, ..., 20$;
- MTrain10(i), MVal10(i), PTrain10(i), and PVal10(i), for $i = 1, ..., 10$;
- MTrain20(i), MVal20(i), PTrain20(i), and PVal20(i), for $i = 1, ..., 5$;
- MTrain25(i), MVal25(i), PTrain25(i), and PVal25(i), for $i = 1, ..., 4$;
- MTrain50(i), MVal50(i), PTrain50(i), and PVal50(i), for $i = 1, 2$.

Prior to each simulation trial, the prior probabilities SCFG are initialized 10 different ways: one in a uniform way, and the remainder in a random way. During each trial with a given size of block, batch learning with the igEM and oigEM techniques is performed on MTrain(1)[1] (PTrain(1)) using 10-fold cross-validation, until the difference between the approximate negative log likelihoods of the sequences on MVal (PVal) is lower than

[1] For simplification, in this subsection, MTrain then refers either to MTrain5, MTrain10, MTrain20, MTrain25 or MTrain50. This simplification also applies to MVal, PTrain, and PVal.

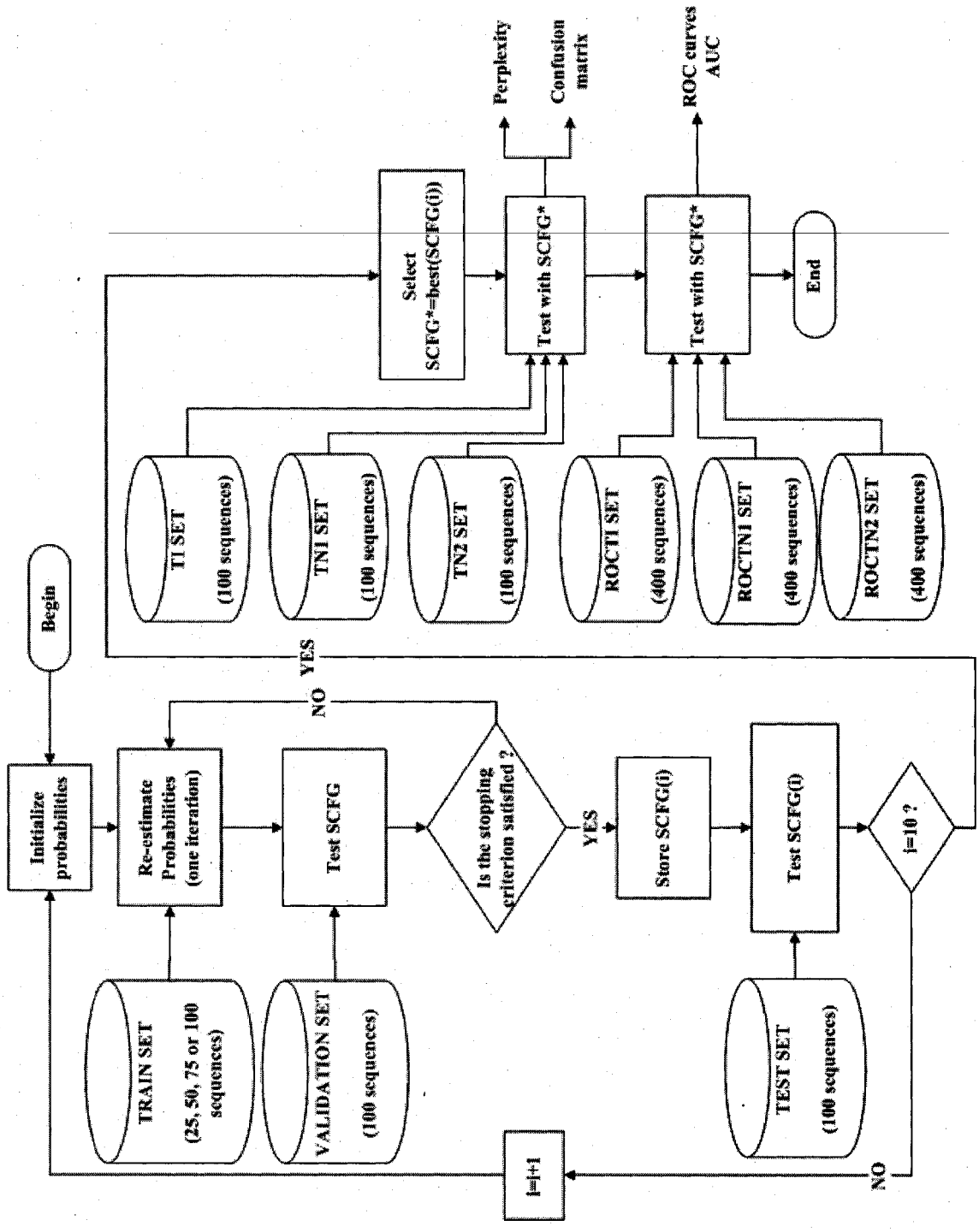


Figure 17 Data flow diagram representation of the experimental protocol used to evaluate the performance of learning techniques in the first part of this thesis.

0.001 for two consecutive iterations. At the end of each trial, two sets of SCFG probabilities are kept, the sets corresponding to the minima of the negative log likelihoods, and of the approximate negative log likelihoods of the words of `MVal` (`PVal`). Finally, the performance on the first block is assessed using `MTest` (`PTest`), and the vectors of sufficient statistics corresponding to each result are stored.

Once `MTrain(1)` (`PTrain(1)`) has been learned via batch learning, `MTrain(2)` (`PTrain(2)`) is learned incrementally. For incremental learning with igEM and oigEM, 10 new sets of SCFG probabilities are generated randomly. For incremental learning with HOLA, the 10 sets of SCFG probabilities from the previous iteration are used (see Section 3.3). Reestimation of the probabilities is performed for each set of probabilities based on the vectors of sufficient statistics stored after learning `MTrain(1)` (`PTrain(1)`). The same process is repeated for successive blocks of training data, until 100 sequences have finally been presented to system. Average results, with corresponding standard error, are always obtained, as a result of the 10 independent simulation trials. To simplify computer simulations, perplexity and recognition of radar states are assessed using `MTI`, $\texttt{MTN}_1$ and $\texttt{MTN}_2$ (`PTI`, $\texttt{PTN}_1$ and $\texttt{PTN}_2$), with the best set of SCFG probabilities once the last block has been learned. Finally, the capacity to recognize the MFR corresponding to the trained grammar is assessed using `MROCTI`, $\texttt{MROCTN}_1$ and $\texttt{MROCTN}_2$ (`PROCTI`, $\texttt{PROCTN}_1$ and $\texttt{PROCTN}_2$).

The diagram of Fig. 17 summarizes the experimental protocol used to assess performance in the second part of this thesis.

5.3 Performance measures

In the present application, the quality of results and the computational efficiency of the learning techniques are equally important. Therefore, two types of performance measures have been considered. To assess the quality of learning techniques, the perplexity measurement, the classification rate over radar states, and ROC curves have been considered.

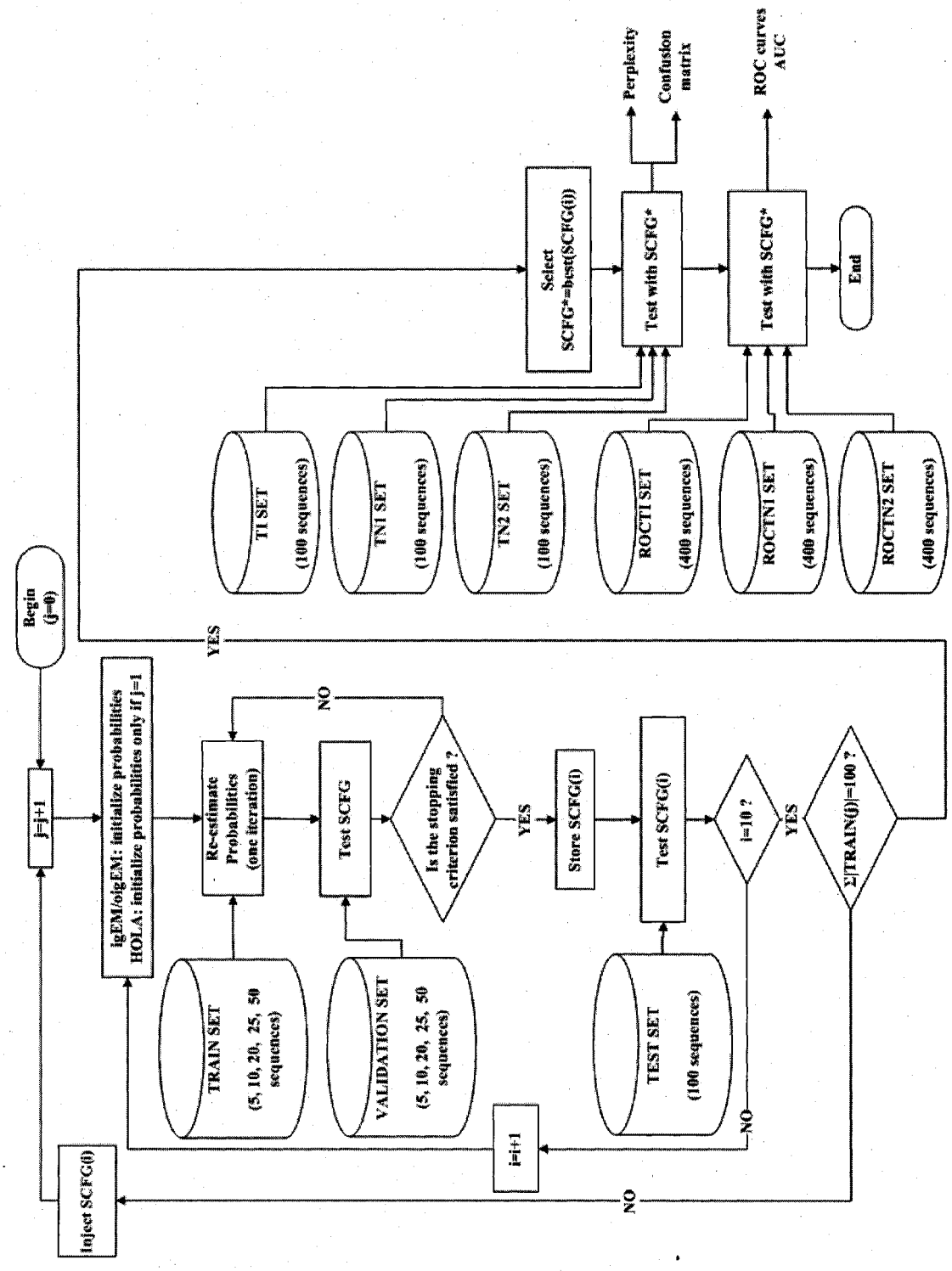


Figure 18 Data flow diagram representation of the experimental protocol used to evaluate the performance of learning techniques in the second part of this thesis.

In contrast, to assess the amount of resources required to implement a technique, the computational complexity, the memory requirements, and the convergence time have been considered.

Estimation of the worst-case and average-case running time per iteration and storage requirements allows for the assessment of computational complexity, whereas the remaining performance measures are assessed via computer simulation. The outcome of numerous computer simulations are combined to yield an average performance measure. Simulation results and complexity estimates are analyzed with ES applications in mind. These measures are now defined in in more detail.

5.3.1 Log-likelihood (LL) and perplexity (PP)

The initial performance measure considered was the negative log likelihood. However this measure depends from the size of the sequences, and therefore is not suitable to compare training on different sizes of phrases. Perplexity(Estève, 2002) is a more objective measure to compare grammars, as it does not depend from the size of the sequence.

According to theory of information, considering a sources producing emissions σ_t, the quantity of information linked to this emission can be defined as:

$$I(\sigma_t) = -\log_2 P(\sigma_t) \tag{5.1}$$

where $P(\sigma_t)$ is the probability of the emission.

This value is positive or null and quantifies the decrease of uncertainty an emission σ_t has drawn from the source. An emission of low probability gives more information than an emission of high probability. An emission with a probability of 1 does not give any information and correspond to a value of 0. Thus the mean value of the quantity of information emitted by a source s is known as the entropy:

$$H(s) = -\sum_{t=1}^{M} P(\sigma_t) \log_2 P(\sigma_t) \tag{5.2}$$

where M is the number of possible values for σ_t.

In radar ES applications, an emission σ_t can correspond to a word w_t. For a sequence of words $x = w_1^n = w_1, ..., w_n$ produced by the source s, the entropy of s becomes:

$$H(s) = \lim_{n \to \text{inf}} -\frac{1}{n} \sum_{t=1}^{n} P(w_t^n) log_2 P(w_t^n) \tag{5.3}$$

where $P(w_1^n)$ is the emission probability of the sequence of words, and $P(w_t^n)$ is the emission probability of the sub-sequence $w_t, ..., w_n$.

In the reality one cannot however observe all the possible sequences produced by a particular source. Therefore, the notion of ergodism has to be introduced. A source is said to be ergodic if every sequence produced by this source is representative of this source that is long enough and allows studying its statistic structure. Thus, we can define the quantity:

$$LP = -\frac{1}{n} \log_2 P(w_1^n) \tag{5.4}$$

where $P(w_1^n)$ is the probability of the sequence $w_1 \dots w_n$ according to a language. LP is an approximation of the entropy defined in Eq. 5.3. Then the perplexity of this language is:

$$PP = 2^{LP} \tag{5.5}$$

The result given by the perplexity can be interpreted as the number of words a model can choose from to complete a given sequence. The lower this value is, the more the model can predict the language.

When applied roughly, this measure has the major disadvantage of giving an infinite value if a sequence has a null probability of occurence, even if this problem only comes from one erroneous word. Therefore the trained grammars have to undergo smoothing, in order to avoid any null probability. There are several techniques to smooth a grammar (Jauvin, 2003). In this thesis, a non-terminal that can produce any combination of non-terminals (including itself) and be produced in any combination by any other non-terminal is simply added, and the corresponding probabilities are set to 10^{-6}. This smoothed grammar is only used punctually during parsing when the perfect one does not allow to fill a cell of the CYK chart or an Earley step. This ensures that the path of maximum probability computed thanks to the Viterbi algorithm will not be affected in case of a correct sequence of words, while eliminating every null probability of a sequence.

Here is also defined the *approximate perplexity* ($\widehat{PP}$), as computed using the maximum likelihood instead of the global likelihood. The formula becomes:

$$\widehat{PP} = 2^{-\frac{1}{L}\log_2 \hat{P}(x)} \tag{5.6}$$

where $\hat{P}(x) = \max_{d_x} P(x, d_x)$ (Eq. 2.4).

5.3.2 Measures of dispersion

To represent a distribution, two criteria are usually used: the mean E and the variance S^2, which comes from the deviation $d = (x - E)$ of each point of the sample. The variance of a sample is expressed as in Eq.5.7.

$$\begin{aligned} S^2 &= \frac{\sum d^2}{n-1} \\ &= \frac{\sum (x - \frac{\sum x}{n})^2}{n-1} \\ &= \frac{\sum x^2 - 2\sum (x \sum \frac{x}{n}) + \sum (\sum \frac{x}{n})^2}{n-1} \end{aligned}$$

$$= \frac{\sum x^2 - 2(\sum x)(\frac{\sum x}{n}) + n(\frac{\sum x}{n})^2}{n-1}$$
$$= \frac{\sum x^2 - \frac{(\sum x)^2}{n}}{n-1} \tag{5.7}$$

where $\{x\}$ is the sample and $n = |\{x\}|$. It can be seen that the denominator of the fraction is $n-1$ and not n. The reason is the following. Usually the sample variance is used to express the variance of a whole distribution. If n is the size of the sample, it means that knowing the mean and the derivation of $n-1$ points the last point can be computed. $n-1$ is called the number of degrees of freedom of the data.

Knowing S^2, S can be computed and is called the standard deviation of the data. But according to the sample, the mean and the standard deviation will change. So a measure that indicates how exact the mean can be necessary. The central limit theorem states that taking enough samples, the distribution of the mean will be a gaussian and it can be proved that its variance will be $\frac{S^2}{n}$. Thus the standard deviation of the mean or standard error of the mean is expressed as $\frac{S}{\sqrt{n}}$.

5.3.3 Classification rate over radar states

One of the important functions of a radar ES system is to estimate the level of threat associated with the state of a MFR. Classification rate over radar states is estimated with the ratio of correctly classified patterns over all test set patterns. As the grammar is designed according to the radar functioning, each non-terminal following the start symbol corresponds to the state of the radar generating the corresponding sequence of four words. These non-terminals are accessible via to the Viterbi parsing algorithm. From the result, it is straightforward to construct a confusion matrix that will display the classification rate over radar states.

A cost analysis of error has not been explored in this thesis, but could be derived from such a confusion matrix. In radar ES, certain errors, can constitute different threat levels. For instance, estimating a `S` state rather than a `Tm` state is much more costly than estimating an `Acq` State rather than a `S` state.

5.3.4 ROC curves

Another important function of radar ES is to recognize the type of MFR system associated with pulse streams. Since a SCFG can be seen as a one-class classifier, this function consists in detecting a MFR with an approximate threshold. Receiver Operating Characteristics (ROC) curves (Fawcett, 2003) are a powerful tool to determine the best threshold and to assess the capacity of the trained grammars to perform MFR recognition.

Having sequences from different MFRs and a SCFG corresponding to a specific MFR, each sequence can be considered as a *Positive* if it belongs to this specific MFR, or as a *Negative* if it belongs to any other MFR. Thus, by computing the corresponding perplexities according to the SCFG and setting a threshold, it allows classifying each sequence: four results can then be extracted – *True Positive* and *False Negative* for correctly and badly classified Positives, and *True Negative* and *False Positive* for correctly and badly classified Negatives. The *True Positive Rate* – or *Hit Rate* (HR) – is then defined as the ratios of the number of True Positives to the total number of Positives. The *False positive rate* – or *False Alarm Rate* (FAR) – is defined as the ratios of the number of False Positives to the total number of Negatives. Creating ROC curves simply consists in making the threshold vary and plotting the corresponding HR with respect to FAR. The best threshold corresponds to the point of the curve closest to (0,1).

The area under the ROC curve (AUC) allows reducing the performance measure from a two dimensional representation to a coarse single scalar value, in order to characterize the ability of a SCFG to recognize the corresponding MFR system. The worse result corresponds to an AUC of 0.5, which means that the SCFG is not able to discriminate its

corresponding MFR from the others. The best result corresponds to an AUC of 1, which means that the SCFG can perfectly discriminate its corresponding MFR from the others. The difference between HR and FAR can also be used.

5.3.5 Convergence time

During a computer simulation, each complete presentation of all the sequences in the training set is called an iteration. Convergence time is measured by counting the number of iterations needed for a leaning technique to converge. Once convergence is reached, perplexity values remain constant two successive presentations of the training set.

5.3.6 Time complexity per iteration

A first order approximation of the computational complexity for the learning techniques for may be obtained by assessing their execution time on an idealized computer. Thus, the time complexity, T, combined with a fixed computing area C, allows for comparison of area-time complexities, CT. To that effect, assume that all algorithms are implemented as computer programs running on a generic, single processor, random access machine (RAM) (Corman *et al.*, 1990), where instructions are executed in sequence. This generic machine is capable of no more than one operation per cycle.

Time complexity can be estimated analytically from the maximum execution time required for one iteration, to re-estimate production rule probabilities of a SCFG based on all the training sequences. The *worst-case* complexity represents the case in which all the possible productions are possible. The result is a total worst-case running time formula, T, which summarizes the behavior of an algorithm as a function of key parameters, such as L and N. In addition, the *average-case* complexity may be estimated by introducing the stochastic parameters such as $|\Delta e|$ in gEM, the number of subgraphs corresponding to an emission, that may be specific to an algorithm.

Specifically, T can be defined as the sum of the worst-case or average-case running times T_p for each operation p that is required to process an input (Corman *et al.*, 1990):

$$T = \sum_p T_p = \sum_p o_p \cdot n_p \tag{5.8}$$

where o_p is the constant amount of time needed to execute an operation p, and n_p is the number of times this operation is executed. Finally, the *growth rate* is obtained by making the parameters of the worst-case complexity tend to ∞.

For simplicity, only the most costly type of operation is considered. We assume that p can take only one value, and o_1 is the time needed to compute a division x/y, a multiplication $x \cdot y$, a square root $\sqrt{x}$, or an exponent e^x. Time complexities are estimated as the number of multiplications, divisions, etc., that occur in one iteration for one sequence, to re-estimate production rule probabilities of a SCFG. In addition, x and y are assumed to be real numbers represented by an integer with a b bit resolution, where b corresponds to the number of bits needed to represent an elementary values (e.g., parameters) with a sufficient precision.

This measure is independent from convergence time, since an algorithm may require several iterations to converge using very simple processing, or vice-versa. The product of the two measures provides useful insight into the amount of processing required by technique to produce its best asymptotic performance. The overall time complexity T^* associated with a learning technique is obtained according to:

$$T^* = T_{\text{init}} + I \cdot T \cdot \Gamma_T \tag{5.9}$$

where T_{init} is the time required initially to produce data structure (prior to iterations), I is the number of iterations needed to converge, T is the time complexity per iteration, and

Γ_T is a multiplicative factor that depends on the size of the dataset $|\Omega|$. It indicates the number of sequences a technique has to treat during the iterative process. Therefore, for batch training, it will usually be equal to $|\Omega|$ (for the EM-based techniques), but it will vary with the techniques when considering incremental learning.

5.3.7 Memory complexity

Memory complexity is estimated as the number of 8 bit registers needed during learning process to store variables. Only the worst-case or average-case memory space required during pre-processing phase was considered. (The temporary memory space required during the iterative processes was neglected.) For gEM, as described in Section 3.2.2, one branch of a support graph consists in 3 vectors of 3 registers (for example $[Start, Acq, \epsilon]$; $[Acq, 0, 4]$; $[\epsilon, 4, 5]$, which means that the non-terminal $Start$ is expanded by Acq ϵ to produce the subsequence $\{w_1 ... w_5\}$), or 2 vectors of 2 and 3 registers ($[W6, 6]$; $[6, 0, 1]$, which means that $W6$ is expanded by 6 to produce w_1), as shown in Fig. 11. With TS, a production rule consists only in a vector of 2 or 3 registers (for example $[W6, 6]$ or $[Start, Acq, \epsilon]$ for emission and transition rules). With HOLA only one register is needed for each rule (to representing its frequency).

This measure contains limited information, due to the fact that only one sequence is considered. In order to measure the impact of incremental learning on memory requirements, as for the time complexity, the overall memory complexity M^* can be defined:

$$M^* = M \cdot \Gamma_M \tag{5.10}$$

where M is the basic memory complexity, and Γ_M is a multiplicative factor that depend on the size of the dataset $|\Omega|$. For instance, for the EM-based techniques, it indicates the number of sequences a technique has to store during the iterative process. Therefore, for batch training, it will usually be equal to $|\Omega|$ (for the EM-based techniques), but it will vary with the techniques when considering incremental learning.

CHAPTER 6

RESULTS AND DISCUSSIONS

In this chapter, the performance of the techniques for learning transition rule probabilities on MFR data are presented and discussed. Section 6.1 presents the results corresponding to the first part of this thesis, which involves batch learning of probabilities using the TS(IO), TS(VS), gEM(IO), gEM(VS) and HOLA algorithms. Then, Section 6.2 presents the results that correspond to the second part of this thesis, which involves characterizing SCFGs obtained after incremental learning using the igEM, oigEM and HOLA algorithms. Performance is assessed on the quality (*i.e.*, the perplexity and classification rate over MFR states), and on the resource requirements (*i.e.*, the convergence time, the time complexity – both overall and per iteration – and the memory complexity – both overall and per iteration) associated with learning techniques.

6.1 Comparison of fast batch learning techniques

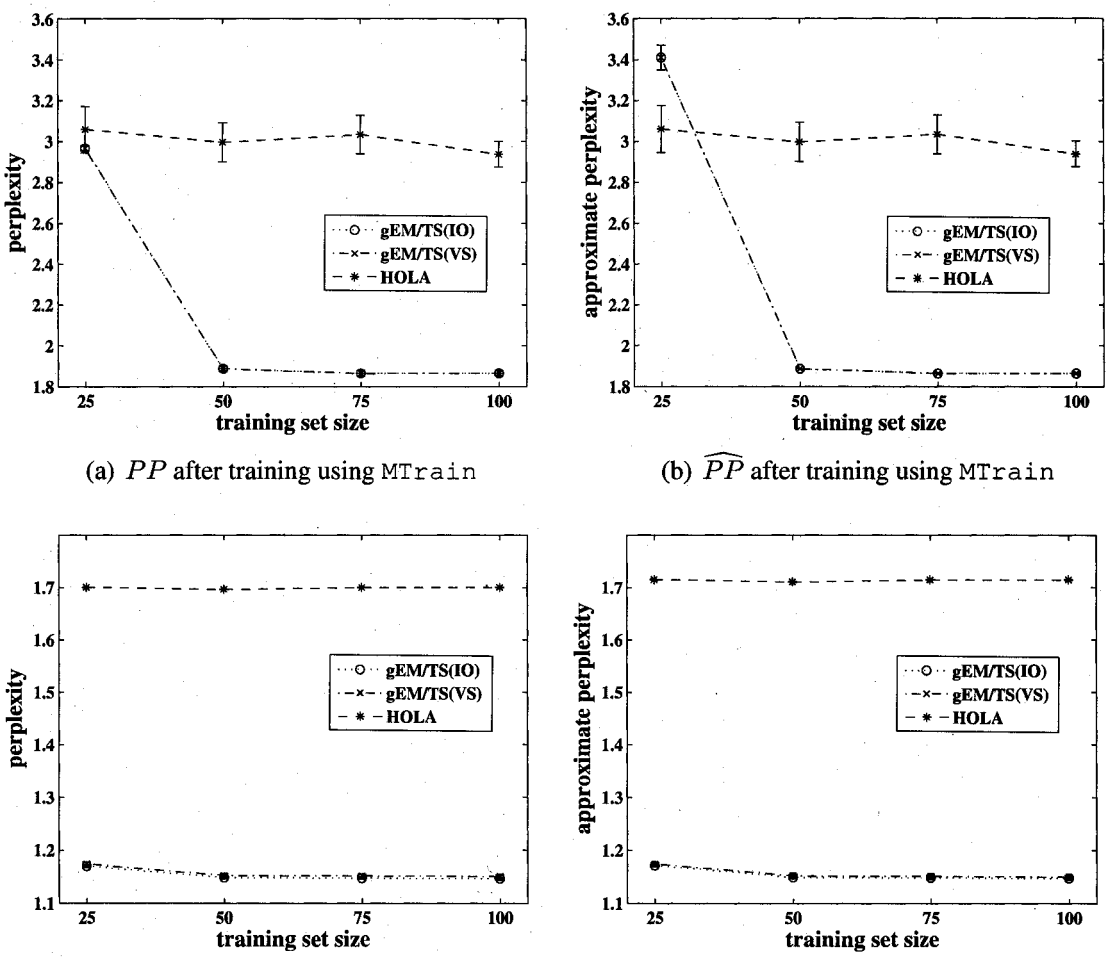
6.1.1 Quality measures

Fig. 19 (a) and (b) shows the average perplexity and approximate perplexity achieved by gEM(IO), gEM(VS), TS(IO), TS(VS) and HOLA on `MTest`, as a function of `MTrain` size. Results for gEM and TS are mathematically equivalent (due to the considerable length of the sequences and the difference of implementations, they are numerically slightly different; however, since this difference is very small, their results are combined in the following).When 50 sequences are used for training, an average perplexity of about 1.9 is obtained when learning with gEM/TS(IO) and gEM/TS(VS), and of about 3.0 with HOLA. The standard error of the mean of results with HOLA over the 10 initializations is about 10^4 times greater than for the other algorithms. While 50 training sequences are required to obtain the lowest perplexity with gEM/TS(IO) and gEM/TS(VS), HOLA

gives approximately the same results whatever the size of `MTrain`. The gEM/TS(IO) and gEM/TS(VS) algorithms give similar results because of the low level of ambiguity of the Mercury language, and the limited computer precision. Indeed, if only one parse tree leads to a sequence, they will give exactly the same results. The difference in performance should emerge as the number of parse trees grows. The high perplexities obtained with the HOLA algorithm may be explained by the fact that, during learning, it does not optimize the perplexity, but instead the relative entropy.

Fig. 19 (c) and (d) shows the average perplexity and approximate perplexity achieved by gEM/TS(IO), gEM/TS(VS) and HOLA on `PTest`, as a function of `PTrain` size. An average perplexity of about 1.2 is obtained when learning with gEM/TS(IO) and gEM/TS(VS), and of about 1.7 with HOLA. As expected, HOLA gives lower accuracy in terms of perplexity, whatever the grammar for this kind of application. As with Mercury data, gEM/TS requires 50 iterations to reach its lower perplexity, while HOLA is not affected by the number of sequences in `PTrain`. However, gEM/TS achieve very low perplexity for fewer training sequences. This may be explained by the fact that Pluto language is less complex than that of Mercury. Therefore, less sequences are needed to be representative of the whole language. In other words, sequences emitted by the Pluto MFR are more similar to each other than sequences emitted by the Mercury MFR. That is why the standard error of the mean is greater for experiments on Pluto data than on Mercury data when HOLA is used for training.

It can be seen that although Pluto's grammar is more ambiguous than Mercury's, it consistently leads to lower perplexity values. Actually, ambiguity represents the number of possible different parse trees that can generate a sequence, while the perplexity represents the predictability of a sequence (as mentioned in Section 5.3). In other words, given the sequence and the operation of a MFR, it represents the number of words that could be suitably added at the end of the sequence. A high level of ambiguity in a grammar does



(a) PP after training using `MTrain`

(b) $\widehat{PP}$ after training using `MTrain`

(c) PP after training using `PTrain`

(d) $\widehat{PP}$ after training using `PTrain`

Figure 19 Average perplexity (PP) and approximate perplexity ($\widehat{PP}$) of SCFGs trained with for gEM/TS(IO), gEM/TS(VS) and HOLA versus `MTrain` and `PTrain` size. Error bars are standard error of the sample mean.

not necessarily prevent it from being a good predictor for the corresponding language. For example, given a part of a sequence to complete, several different derivation trees (belonging to an ambiguous grammar) can lead to the same completion. In this case, Pluto has the more ambiguous grammar, but Mercury has a more perplexing and less predictable language.

Fig. 20 displays the average convergence time corresponding to the results found in Fig. 19. With the Mercury data (See Fig. 20 (a)), both gEM/TS(IO) and gEM/TS(VS) require very few iterations to converge (about 4 iterations on average), regardless of the training set size, while HOLA requires between 75 and 95 iterations to converge. With Pluto data (See Fig. 20 (b)), it can be seen that gEM/TS(VS) converges about 4 times faster than gEM/TS(IO), and that gEM/TS(IO) requires more iterations to converge than when using the Mercury data. This is due to the difference in the level of ambiguity of the grammars. Since VS versions of the algorithms optimize only one parse tree, if the best parse tree remains the same in two iterations, the algorithms converge rapidly, which explains the low number of iterations for both grammars. In contrast, since Mercury has a low-ambiguity language, it appears to make the IO-based algorithms converge more rapidly. Results suggest that the convergence time of gEM/TS(IO) is more sensitive to the grammar ambiguity than gEM(VS). Moreover, since HOLA computes histograms to re-estimate the SCFG probabilities, the size of the training database does not have impact on the associated convergence time, whatever the MFR considered. Indeed, it only depends on the number of rules of the associated SCFG. Therefore, since the Pluto grammar has a lower number of production rules than the Mercury grammar, HOLA requires about 14 times fewer iterations for Pluto data than for the Mercury data. The differences between the convergence times for a same MFR result from the heuristic aspect of HOLA.

From these results, it appears that about 50 training sequences generally correspond to the least amount of resources (computational time and memory complexity) that yields the lowest perplexity values. The SCFG corresponding to the lowest perplexity on `MTest` and `PTest` across the 10 initializations, when `MTrain` and `PTrain` sizes are 50, was selected for gEM, TS, and HOLA. In the following discussion, we will refer to these SCFGs as the "best" SCFGs.

Tables IV and V display the confusion matrices obtained by the "best" SCFGs on the `MTI/PTI`, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$ datasets. They present the classification of

Table IV

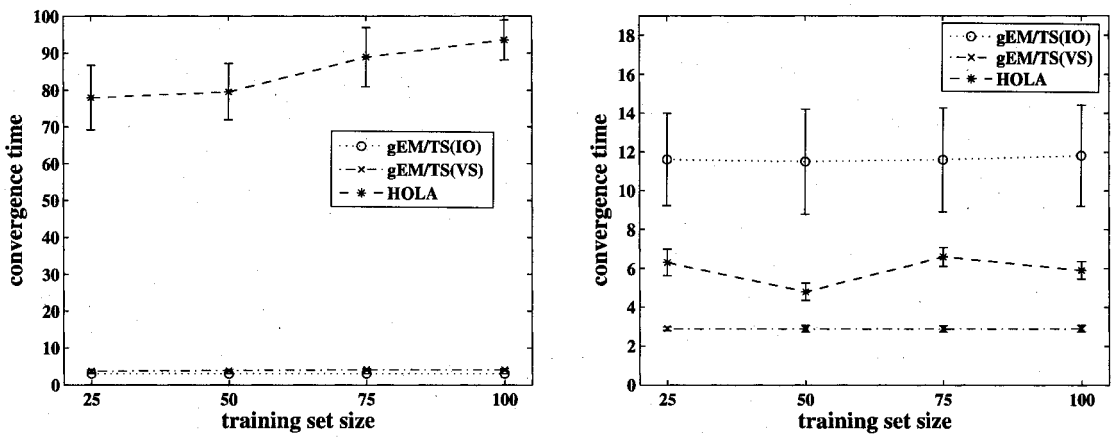
Confusion matrix associated with PTI, PTN_1, and PTN_2 for the "best" SCFGs obtained after learning of PTrain with gEM(IO), gEM(VS) and HOLA using the 50 first sequences.

			Estimated States					Recognition rate per state
			S TI/TN_1/TN_2	Acq TI/TN_1/TN_2	Na TI/TN_1/TN_2	Rr TI/TN_1/TN_2	Tm TI/TN_1/TN_2	
Real States	S	gEM/TS(IO)	4178 / 3500 / 2433	0 / 107 / 343	0 / 150 / 164	0 / 18 / 110	0 / 66 / 341	**1 / 0.911 / 0.717**
		gEM/TS(VS)	4176 / 3641 / 2433	0 / 101 / 336	0 / 174 / 165	0 / 35 / 109	0 / 66 / 342	**1 / 0.906 / 0.719**
		HOLA	4178 / 3380 / 2052	0 / 134 / 377	0 / 161 / 255	0 / 37 / 217	0 / 146 / 569	**1 / 0.846 / 0.591**
	Acq	gEM/TS(IO)	0 / 10 / 22	1734 / 1530 / 1044	235 / 345 / 342	0 / 7 / 66	0 / 0 / 85	**0.88 / 0.804 / 0.67**
		gEM/TS(VS)	0 / 10 / 20	1734 / 1534 / 1045	235 / 331 / 342	0 / 6 / 65	0 / 11 / 86	**0.88 / 0.811 / 0.671**
		HOLA	0 / 3 / 35	1734 / 1452 / 800	235 / 355 / 281	0 / 25 / 92	0 / 52 / 213	**0.88 / 0.769 / 0.563**
	Na	gEM/TS(IO)	0 / 0 / 0	0 / 8 / 7	736 / 613 / 412	0 / 77 / 212	0 / 0 / 34	**1 / 0.878 / 0.62**
		gEM/TS(VS)	0 / 0 / 0	0 / 8 / 5	736 / 615 / 411	0 / 76 / 215	0 / 0 / 33	**1 / 0.88 / 0.619**
		HOLA	0 / 0 / 9	0 / 11 / 8	736 / 571 / 315	0 / 71 / 167	0 / 47 / 135	**1 / 0.816 / 0.497**
	Rr	gEM/TS(IO)	0 / 31 / 21	0 / 0 / 16	0 / 51 / 122	2073 / 1852 / 1267	0 / 55 / 288	**1 / 0.931 / 0.739**
		gEM/TS(VS)	0 / 31 / 24	0 / 0 / 15	0 / 64 / 111	2073 / 1837 / 1270	0 / 54 / 300	**1 / 0.925 / 0.739**
		HOLA	0 / 31 / 66	0 / 0 / 17	0 / 39 / 165	2073 / 1783 / 958	0 / 115 / 403	**1 / 0.906 / 0.595**
	Tm	gEM/TS(IO)	0 / 445 / 1196	0 / 68 / 118	0 / 364 / 523	0 / 74 / 179	6243 / 5389 / 4249	**1 / 0.85 / 0.678**
		gEM/TS(VS)	0 / 483 / 1197	0 / 69 / 118	0 / 316 / 508	0 / 74 / 177	6243 / 5398 / 4265	**1 / 0.851 / 0.681**
		HOLA	0 / 552 / 1284	0 / 116 / 259	0 / 415 / 548	0 / 135 / 455	6243 / 5122 / 3716	**1 / 0.808 / 0.593**
Confidence on estimated state		**gEM/TS(IO)**	**1 / 0.878 / 0.663**	**1 / 0.893 / 0.683**	**0.758 / 0.402 / 0.264**	**1 / 0.913 / 0.691**	**1 / 0.978 / 0.85**	
		gEM/TS(VS)	**1 / 0.874 / 0.662**	**1 / 0.896 / 0.688**	**0.758 / 0.41 / 0.267**	**1 / 0.906 / 0.692**	**1 / 0.976 / 0.849**	
		HOLA	**1 / 0.852 / 0.595**	**1 / 0.848 / 0.548**	**0.758 / 0.31 / 0.201**	**1 / 0.869 / 0.507**	**1 / 0.934 / 0.738**	

Table V

Confusion matrix associated with PTI, PTN_1, and PTN_2 for the "best" SCFGs obtained after learning of PTrain with gEM(IO), gEM(VS) and HOLA using the first 50 sequences.

Real States			Estimated States: S (TI / TN_1 / TN_2)	Estimated States: Na (TI / TN_1 / TN_2)	Estimated States: Rr (TI / TN_1 / TN_2)	Estimated States: Tm (TI / TN_1 / TN_2)	Recognition rate per state
S		gEM/TS(IO)	9041 / 13064 / 9827	5517 / 2576 / 1774	0 / 9 / 1053	1582 / 123 / 1599	**0.56 / 0.828 / 0.689**
		gEM/TS(VS)	9041 / 13067 / 9799	5517 / 2570 / 1796	0 / 9 / 1054	1582 / 123 / 1603	**0.56 / 0.829 / 0.688**
		HOLA	183 / 48 / 791	14375 / 15347 / 8166	0 / 65 / 1124	1582 / 345 / 4271	**0.011 / 0.003 / 0.055**
Acq		gEM/TS(IO)	0 / 326 / 325	1845 / 1149 / 266	0 / 316 / 798	0 / 0 / 334	**1 / 0.642 / 0.154**
		gEM/TS(VS)	0 / 325 / 327	1845 / 1148 / 264	0 / 317 / 798	0 / 0 / 335	**1 / 0.641 / 0.258**
		HOLA	0 / 6 / 49	1845 / 1470 / 483	0 / 310 / 823	0 / 11 / 383	**1 / 0.818 / 0.278**
Na		gEM/TS(IO)	0 / 66 / 315	0 / 0 / 18	2016 / 1645 / 443	0 / 268 / 1049	**1 / 0.831 / 0.243**
		gEM/TS(VS)	0 / 66 / 315	0 / 0 / 18	2016 / 1644 / 441	0 / 269 / 1051	**1 / 0.831 / 0.242**
		HOLA	0 / 20 / 25	0 / 41 / 216	2016 / 1669 / 527	0 / 252 / 1068	**1 / 0.842 / 0.287**
Tm		gEM/TS(IO)	2685 / 3246 / 4061	0 / 142 / 480	0 / 0 / 446	8928 / 8025 / 5504	**0.769 / 0.703 / 0.525**
		gEM/TS(VS)	2685 / 3252 / 4070	0 / 142 / 481	0 / 0 / 446	8928 / 8017 / 5493	**0.769 / 0.703 / 0.524**
		HOLA	17 / 106 / 374	2668 / 3160 / 2831	0 / 8 / 419	8928 / 8152 / 6905	**0.769 / 0.713 / 0.656**
Confidence on estimated state		**gEM/TS(IO)**	**0.771 / 0.782 / 0.25**	**0.749 / 0.297 / 0.105**	**1 / 0.835 / 0.162**	**0.85 / 0.954 / 0.649**	
		gEM/TS(VS)	**0.771 / 0.782 / 0.25**	**0.749 / 0.297 / 0.103**	**1 / 0.835 / 0.161**	**0.85 / 0.953 / 0.468**	
		HOLA	**0.915 / 0.267 / 0.638**	**0.098 / 0.073 / 0.041**	**1 / 0.813 / 0.182**	**0.85 / 0.931 / 0.547**	



(a) Mercury data

(b) Pluto data

Figure 20 Average convergence time required by the gEM/TS(IO), gEM/TS(VS) and HOLA algorithms, versus `MTrain` and `PTrain` sizes.

words from the test sequences according to the different states of a MFR. In fact, they compare the states from the derivation of minimum approximate perplexity of each sequence with the corresponding real states. Although the SCFG obtained with HOLA usually produced the greater number of misclassification, all SCFGs behave consistently. It can be seen that the VS versions of the algorithms gives almost the same results as the IO versions. In Table IV, when `MTI` is processed, the estimation errors only occur when `Na` is estimated instead of `Acq`. Far more errors occur in Table V. This confirms the earlier interpretation of ambiguity for MFRs (see Section 5.1), since classification errors can only appear when several parse trees exist for the same sequence. For the ideal Mercury grammar for the `MTI`, ambiguity only comes from the fact that `Acq` and `Na` can produce the same phrase, while there is more tolerance in the smoothed grammars used for $\texttt{MTN}_1$ and $\texttt{MTN}_2$.

The ability of SCFGs to estimate radar states degrades with the proportion of noisy data. The last rows and columns of the tables represent the classification rate per state, from two different points of view. The last row indicates the confidence that can be associated with an estimated state – that is, the ratio of corrected estimated state on the total number

of times this states was estimated. The last column indicates the ability to recognize the real states – that is, the ratio of corrected estimated state on the total number of times this states really appears. In an ES application, the worst-case situation consists in recognizing a S state instead of a Tm state. The SCFGs often estimate a Na state instead of another one. This means that less confidence should be given when such a state is estimated.

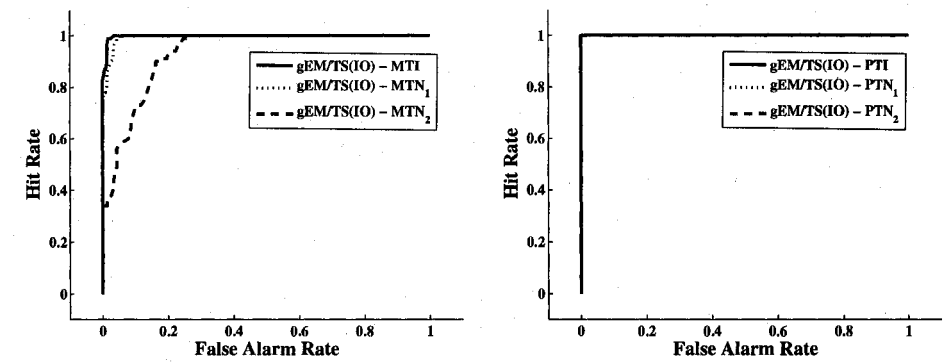
When noise is injected into Mercury test data (MTN_1 and MTN_2), the greatest numbers of errors are perceived when S or Na are estimated instead of Tm. When noise is injected into Pluto test data (MTN_1 and MTN_2), the greatest numbers of errors are perceived when S or Na are estimated instead of Tm, and inversely. Except for the confusion between S and Tm with Mercury data, this is caused when consecutive states share same phrases. The confusion between S and Tm with Mercury data is probably caused by the fact that Tm is a very long state, and that the probability of passing from Tm to S is very low. This table could allow modifying the SCFGs probabilities by hand in order to avoid very dangerous errors, such as estimating S instead of Tm.

Table VI

Perplexity of SCFGs obtained with gEM/TS(IO), gEM/TS(VS) and HOLA on MTI/PTI, MTN_1/PTN_1, and MTN_2/PTN_2.

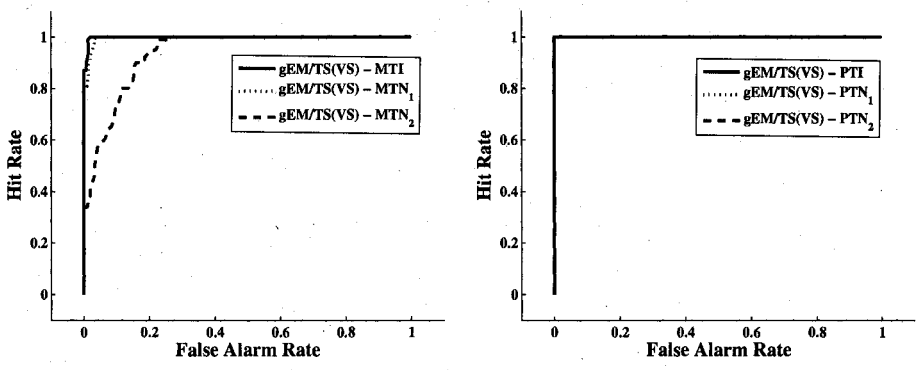
	MERCURY			PLUTO		
Test subset	**gEM/TS(IO)**	**gEM/TS(VS)**	**HOLA**	**gEM/TS(IO)**	**gEM/TS(VS)**	**HOLA**
MTI/PTI	1.82	1.89	2.49	1.28	1.28	1.47
MTN_1/PTN_1	3.14	3.12	4.05	1.76	1.76	1.99
MTN_2/PTN_2	9.99	8.79	12.97	4.49	4.49	4.49

Table VI shows the mean perplexity obtained for the "best" SCFGs on the MTI/PTI, MTN_1/PTN_1, and MTN_2/PTN_2 test sets. Although perplexity tends to grow with the level of noise for all three techniques, the perplexity obtained with gEM and TS is always significantly lower than with HOLA. All the algorithms give the same results for PTN_2 (the computations has reached the limit of precision for the computer).



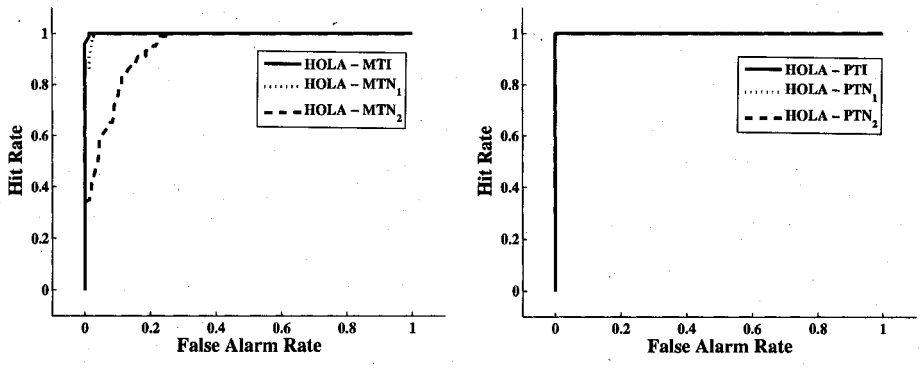
(a) Mercury - gEM/TS(IO)
AUC = 0.998 / 0.994 / 0.931

(b) Pluto - gEM/TS(IO)
AUC = 1 / 1 / 1



(c) Mercury - gEM/TS(VS)
AUC = 0.998 / 0.996 / 0.937

(d) Pluto - gEM/TS(VS)
AUC = 1 / 1 / 1



(e) Mercury - HOLA
AUC = 0.999 / 0.997 / 0.940

(f) Pluto - HOLA
AUC = 1 / 1 / 1

Figure 21 ROC curves obtained for SCFGs trained using (a) gEM/TS(IO); (c) gEM/TS(VS); (e) HOLA on MROCTI, $\texttt{MROCTN}_1$, and $\texttt{MROCTN}_2$ data, and trained using (b) gEM/TS(IO); (d) gEM/TS(VS); (f) HOLA on PROCTI, $\texttt{PROCTN}_1$, and $\texttt{PROCTN}_2$ data. The caption of each subfigure also gives the AUC for $\texttt{MROCTI}/\texttt{MROCTN}_1/\texttt{MROCTN}_2$ and $\texttt{PROCTI}/\texttt{PROCTN}_1/\texttt{PROCTN}_2$.

Fig. 21 shows the ROC curves for the Mercury and Pluto grammars, using the "best" SCFGs on MROCTI, $\texttt{MROCTN}_1$, $\texttt{MROCTN}_2$ data, and trained on and PTrain, using PROCTI, $\texttt{PROCTN}_1$, and $\texttt{PROCTN}_2$ data. Recall that these databases consist of 400 sequences. The first 100 sequences of MROCTI, $\texttt{MROCTN}_1$, and $\texttt{MROCTN}_2$ belong to Mercury language, whereas the last 300 sequences belong to the three other MFRs languages. The first 100 sequences of PROCTI, $\texttt{PROCTN}_1$, and $\texttt{PROCTN}_2$ belong to Pluto language, whereas the last 300 sequences belong to the three other MFRs languages. By computing the perplexity associated with each test sequence and by varying a threshold across the range of perplexity values on can compute the Hit Rate and the False Alarm Rate of a SCFG (see Section 5.3.4).

It can be seen that performance degrades when noise is introduced for the Mercury grammar, but that the SCFGs remains quite discriminant even for a noise frequency of on error in 10 words. Performance for the IO and VS versions of gEM and TS do not give significant difference. Analysis of the detection error show that the detection errors (FAR) come from sequences from VATI and VBTI, because Venus languages appear to be quite close to Mercury's, while Pluto's language is more distinct from the three others. Indeed, these curves show that increasing noise to a frequency of one in 10 words for the replacements does not have a significant impact on the performance obtained with Pluto SCFGs, using either the IO or the VS versions of gEM and TS, or HOLA.

Table VII displays the decision threshold γ^* that corresponds to the best discrimination, along with the corresponding values of HR and FAR, when the difference between HR and FAR is maximized: $\gamma^* = \text{argmax}\{HR(\gamma) - FAR(\gamma)\}$ (note that these operation point do not count for the costs of errors). For results from experiments on Mercury data, as expected, the thresholds are just a little higher than the mean perplexities given in Table VI. For results from experiments on Mercury data, however, the difference is significantly higher. This only means than a few sequences give a higher perplexity, but it can be seen that it does not affect the ability of the model to recognize the MFR.

Table VII

Best decision thresholds, and associated HR and FAR, for SCFGs trained with gEM/TS(IO), gEM/TS(VS), and HOLA on MTI/PTI, $\texttt{MTN}_1$/$\texttt{PTN}_1$, and $\texttt{MTN}_2$/$\texttt{PTN}_2$.

	MERCURY		
	gEM/TS(IO) MTI/$\texttt{MTN}_1$/$\texttt{MTN}_2$	**gEM/TS(VS)** MTI/$\texttt{MTN}_1$/$\texttt{MTN}_2$	**HOLA** MTI/$\texttt{MTN}_1$/$\texttt{MTN}_2$
γ^*	3.84 / 4.54 / 8.50	3.80 / 4.54 / 8.50	5.07 / 5.67 / 11.34
HR	0.99 / 0.99 / 0.99	0.99 / 0.99 / 0.99	0.99 / 0.99 / 0.99
FAR	0.02 / 0.04 / 0.23	0.02 / 0.04 / 0.23	0.01 / 0.04 / 0.23

	PLUTO		
	gEM/TS(IO) MTI/$\texttt{MTN}_1$/$\texttt{MTN}_2$	**gEM/TS(VS)** MTI/$\texttt{MTN}_1$/$\texttt{MTN}_2$	**HOLA** MTI/$\texttt{MTN}_1$/$\texttt{MTN}_2$
γ^*	4.09 / 4.62 / 6.4	4.09/4.62/6.4	4.52/5.04/6.93
HR	1 / 1 / 1	1 / 1 / 1	1 / 1 / 1
FAR	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0

Overall, it appears that the SCFGs trained with gEM/TS(IO), gEM/TS(IO), and HOLA can always detect the MFR of interest with a very high level of accuracy. Of all the situations considered, the lowest hit rate of about 85% and false alarm rate of about 13% is observed for HOLA on $\texttt{MROCTN}_2$. This corresponds to an extreme case, in which only the sequences emitted by the MFR of interest are noisy, and the level of noise is high. Moreover, the lower AUC value is 0.93 (0.99 when not considering tests on $\texttt{MROCTN}_2$).

6.1.2 Resources requirements

Tables VIII and IX present the estimates of time complexity per iteration T and memory complexity M of the learning techniques used for batch training. Recall that T represents the number of main operations (multiplication and divisions) needed for one iteration of training, for one sequence. M represents the number of register needed to store the training data for one iteration and for one sequence. Resources associated with the validation sets are not considered. In these tables:

- $|r|$ is the number of rules of the grammar;
- L is the length of a training sequence;
- M_t is the number of emission rules;
- $|\varphi_t|$ and $|\varphi_e|$ are the average numbers of sub-graphs in support graph corresponding to transition and emission rules;
- $|\Delta l_t|$ is the average number of branches per sub-graph corresponding to a transition rule;
- $|Tr|$ is the average number of trees leading to a sequence;
- $|\Delta Tr|$ is the average size of a tree, that is the average number of production rules per tree;
- $|\Delta Tr_t|$ is the average number of transition rules per tree;
- $|\Delta Tr_e|$ is the average number of emission rules per tree.

In order to compute the worst-case estimations, the following approximations have been considered:

- $|\varphi_t| = M_{nt} \cdot \sum_{i=1}^{L} \sum_{j=1}^{i-1} L = M_{nt} \cdot (L^2 - 1)$
- $|\varphi_e| = M_{nt} \cdot L$
- $|\Delta l_t| = M_{nt}^2 \cdot L$
- $|\Delta Tr| = 2 \cdot L - 1$
- $|Tr| = M_{nt}^L$
- $|\Delta Tr_t| = L - 1$

Table VIII

Estimation of the time complexity per iteration for learning techniques.

	T		
Techniques	**Average**	**Worst-case**	**Growth rate**
Inside-Outside (IO)	$\frac{4}{3}M_{nt}^3 \cdot L \cdot (L^2-1)$ $+M_{nt} \cdot M_t \cdot (L+2)$ $+(L^2+1)+2 \cdot M_{nt}^3$	$\frac{4}{3}M_{nt}^3 \cdot L \cdot (L^2-1)$ $+M_{nt} \cdot M_t \cdot (L+2)$ $+(L^2+1)+2 \cdot M_{nt}^3$	$O(M_{nt}^3 \cdot L^3)$
Viterbi Scores (VS)	$M_{nt}^3 \cdot (\frac{L \cdot (L^2-1)}{3}+1)$	$M_{nt}^3 \cdot (\frac{L \cdot (L^2-1)}{3}+1)$	$O(M_{nt}^3 \cdot L^3)$
gEM(IO)	$6\,\lvert\varphi_e\rvert + 9 \cdot \lvert\varphi_t\rvert \cdot \lvert\Delta l_t\rvert$	$6 \cdot M_{nt} \cdot L+$ $9 \cdot M_{nt}^3 \cdot L \cdot (L^2-1)$	$O(M_{nt}^3 \cdot L^3)$
gEM(VS)	$6 \cdot \lvert\varphi_e\rvert+$ $3 \cdot \lvert\varphi_t\rvert \cdot (\lvert\Delta l_t\rvert+2)$	$6 \cdot M_{nt} \cdot L+$ $3 \cdot M_{nt} \cdot (L^1-1) \cdot (M_{nt}^2+2)$	$O(M_{nt}^3 \cdot L^3)$
TS(IO)	$\lvert\Delta Tr\rvert \cdot \lvert Tr\rvert$	$(2 \cdot L-1) \cdot M_{nt}^L$	$O(M_{nt}^L \cdot L^3)$
TS(VS)	$\lvert\Delta Tr\rvert \cdot \lvert Tr\rvert$	$(2 \cdot L-1) \cdot M_{nt}^L$	$O(M_{nt}^L \cdot L^3)$
HOLA	$2 \cdot \lvert r\rvert$	$2 \cdot (M_{nt}^3 + M_{nt} \cdot M_t)$	$O(M_{nt}^3)$

Table IX

Estimation of the memory complexity for learning techniques.

	M		
Techniques	**Average**	**Worst-case**	**Growth rate**
Inside-Outside (IO)	$2 \cdot L^2 \cdot M_{nt}$	$2 \cdot L^2 \cdot M_{nt}$	$O(L^2 \cdot M_{nt})$
Viterbi Scores (VS)	$2 \cdot L^2 \cdot M_{nt}$	$2 \cdot L^2 \cdot M_{nt}$	$O(L^2 \cdot M_{nt})$
gEM(IO)	$L^2 \cdot M_{nt} \cdot (2+\lvert\Delta_l\rvert)+$ $4 \cdot \lvert\varphi_e\rvert + 9 \cdot \lvert\varphi_t\rvert \cdot \lvert\Delta l_t\rvert$	$2 \cdot M_{nt} \cdot L \cdot (2+L)$ $+M_{nt}^3 \cdot (10 \cdot L^3-8)$	$O(M_{nt}^3 \cdot L^3)$
gEM(VS)	$L^2 \cdot M_{nt} \cdot (2+\lvert\Delta_l\rvert)+$ $4 \cdot \lvert\varphi_e\rvert$ $+\lvert\varphi_t\rvert \cdot (9 \cdot \lvert\Delta l_t\rvert+1)$	$M_{nt}^3 \cdot (l^3+1)+M_{nt} \cdot [3 \cdot L^2$ $+L \cdot (4-3 \cdot M_{nt}^2)$ $+3 \cdot M_{nt}^2 \cdot L^3-1]$	$O(M_{nt}^3 \cdot L^3)$
TS(IO)	$(3 \cdot \lvert\Delta Tr_t\rvert$ $+2 \cdot \lvert\Delta Tr_e\rvert) \cdot \lvert Tr\rvert$	$(5 \cdot L-3) \cdot M_{nt}^L$	$O(M_{nt}^L \cdot L)$
TS(VS)	$(3 \cdot \lvert\Delta Tr_t\rvert$ $+2 \cdot \lvert\Delta Tr_e\rvert) \cdot \lvert Tr\rvert$	$(5 \cdot L-3) \cdot M_{nt}^L$	$O(M_{nt}^L \cdot L)$
HOLA	$4 \cdot \lvert r\rvert$	$2 \cdot (M_{nt}^3 + M_t \cdot M_{nt})$	$O(M_{nt}^3)$

- $\lvert\Delta Tr_e\rvert = L$

For IO and VS, the average and worst-case time and memory complexities are the same, regardless of SCFG. In contrast, the average complexities for gEM, TS and HOLA differ considerably from the average to the worst-case. The time and memory complexities for gEM and TS grow exponentially with the inherent ambiguity of the SCFG. Indeed, for gEM, increasing the ambiguity of a grammar will increase both $|\varphi_t|$ and $|\Delta l_t|$. For TS, it will increase both $|\Delta Tr|$ and $|Tr|$. That is why these algorithms are well adapted for radar ES applications, in which the grammars are less ambiguous than with natural languages analysis, for instance. The time and memory complexities grow linearly with the number of SCFG rules for HOLA. That is why, overall, HOLA has the lowest time and memory complexity of the algorithms. The others may have a very low time complexity but require a substantial amount of memory to store data structures.

The average case T and M values corresponding to the simulation results on Mercury data presented in this thesis may also been computed. These values correspond to the case in which learning algorithms are trained using the first 50 sequences of `MTrain`. In order to compute T and M for one iteration and one sequence, the values obtained after the whole training were normalized by the convergence time and the size of the training database. The values corresponding to IO and VS were computed using the formulas of Tables VIII and IX.

- $T_{HOLA} = 1.76 \cdot 10^2 < T_{TS(IO)} = T_{TS(VS)} = 7.32 \cdot 10^3 < T_{gEM(VS)} \approx T_{gEM(IO)} = 9.78 \cdot 10^3 < T_{VS} = 5.02 \cdot 10^{12} < T_{IO} = 2.64 \cdot 10^{13}$;
- $M_{HOLA} = 3.52 \cdot 10^2 < M_{TS(IO)} = M_{TS(VS)} = 7.35 \cdot 10^3 < M_{IO} = M_{VS} = 2.90 \cdot 10^7 < M_{gEM(IO)} \approx M_{gEM(VS)} \approx 4.36 \cdot 10^7$.

The average case T and M values corresponding to the experiments on Pluto data presented in this section may also been computed.

- $T_{HOLA} = 0.70 \cdot 10^2 < T_{gEM(VS)} = 1.72 \cdot 10^4 < T_{gEM(IO)} = 1.88 \cdot 10^4 < T_{TS(IO)} = T_{TS(VS)} = 6.46 \cdot 10^9 < T_{VS} = 6.28 \cdot 10^{11} < T_{IO} = 2.51 \cdot 10^{12}$;
- $M_{HOLA} = 1.40 \cdot 10^2 < M_{IO} = M_{VS} = 1.49 \cdot 10^7 < M_{gEM(IO)} \approx M_{gEM(VS)} \approx 2.24 \cdot 10^7 < M_{TS(IO)} = M_{TS(VS)} = 1.61 \cdot 10^{10}$.

As expected, HOLA always has the lowest time complexity per iteration and memory complexity. For experiments on Mercury data, TS needs less resources than gEM, for either IO or VS versions. This situation is inverted for experiments on Pluto data.

From these last values, it is now possible to compute the overall average case time $T^* = T_{\text{init}} + T \cdot \Gamma_T \cdot I$ and memory $M^* = M \cdot \Gamma_M$ complexity for TS(IO), TS(VS), gEM(IO), gEM(VS),and HOLA (this is not possible for IO and VS, since their convergence time is unknown here). The influence of the preprocessing, T_{init}, is not considered. For TS(IO), TS(VS), gEM(IO), and gEM(VS), Γ_T and Γ_M corresponds to the number of sequences of the database. Considering that training has been performed on a database of 50 sequences, $\Gamma_T = \Gamma_M = |\Omega| = 50$. For HOLA, the values of Γ_T and Γ_M corresponds to the number of sequences it generates at each iteration. Let's consider that $\Gamma_T = \Gamma_M = |\Omega| = 50$ (see Section 5.3 for details). Accordingly, for the experiments on Mercury data, T^* and M^* are:

- $T^*_{HOLA} = 7.05 \cdot 10^5 < T^*_{TS(IO)} = T^*_{TS(VS)} = 1.10 \cdot 10^6 < T^*_{gEM(IO)} = T^*_{gEM(VS)} = 1.47 \cdot 10^6$;
- $M^*_{HOLA} = 1.76 \cdot 10^4 < M^*_{TS(IO)} = M^*_{TS(VS)} = 3.68 \cdot 10^5 < M^*_{gEM(IO)} \approx M^*_{gEM(VS)} \approx 2.18 \cdot 10^9$.

and for the experiments on Pluto data, T^* and M^* are:

- $T^*_{HOLA} = 3.15 \cdot 10^5 < T^*_{gEM(VS)} = 1.47 \cdot 10^6 < T^*_{TS(VS)} = 1.10 \cdot 10^6 < T^*_{TS(IO)} = 3.88 \cdot 10^{12} < T^*_{gEM(IO)} = 1.13 \cdot 10^7$;

- $M^*_{HOLA} = 7.00 \cdot 10^3 < M^*_{gEM(IO)} \approx M^*_{gEM(VS)} \approx 1.12 \cdot 10^9 < M^*_{TS(IO)} = M^*_{TS(VS)} = 8.05 \cdot 10^{11}$.

Despite having the grater convergence time in experiments on Mercury and Pluto data, HOLA achieves the lower overall time complexity of all the techniques. Since all the algorithms have almost the same convergence time on Mercury data, it can be seen that TS(IO), TS(VS), gEM(IO) and gEM(VS) are ranked in the same order as when only considering the time complexity per iteration. In contrast, since there are some differences in the convergence times between the IO and VS versions of the algorithms for the experiments on Pluto data, TS(VS) has a lower overall time complexity than gEM(IO).

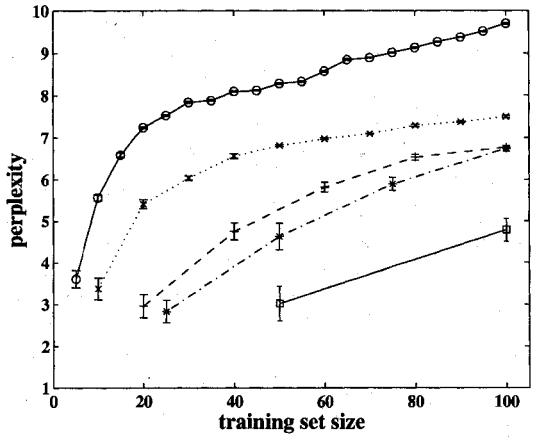
Since the convergence time has no influence on the overall memory complexity M^*, all the techniques are ranked in the same order as when considering memory complexity M.

6.2 Comparison of fast incremental learning techniques

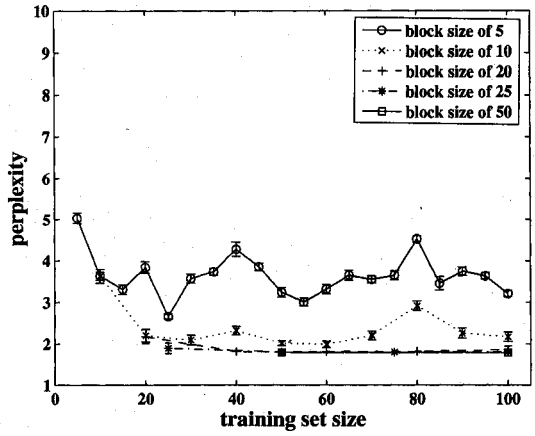
6.2.1 Quality measures

Figs. 22 and 23 show the average perplexity achieved by a SCFG obtained by incremental learning with igEM, oigEM and HOLA for different block sizes $|\Omega_i|$ of the Mercury and Pluto data. Assuming that the environment is static, and that a block Ω_i is representative of the environment, performance depends heavily on its size $|\Omega_i|$. If Ω_i is large enough, the SCFG will be well defined.

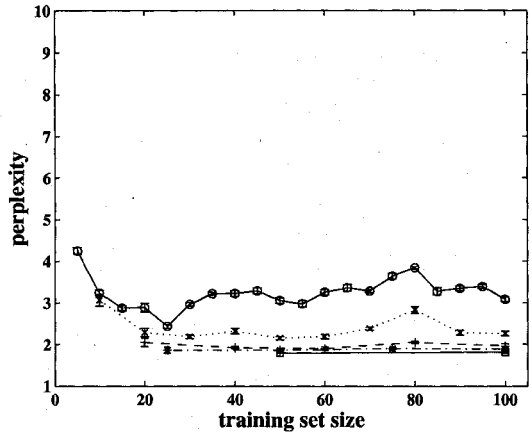
With Mercury data, the perplexity of a SCFG obtained from either igEM or oigEM tends towards the behaviour that could be achieved through batch learning on one single cumulative block of data, even for small block sizes. It appears that $|\Omega_i|$ = 25 sequences is required to reach the lowest possible perplexity. At that point, incremental learning with igEM or oigEM gives the same performance as batch learning with gEM. Note that igEM and oigEM are equivalent to gEM when used for batch learning of a single block of data.



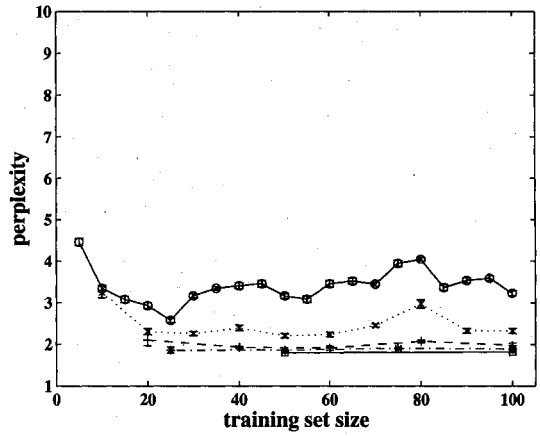
(a) HOLA



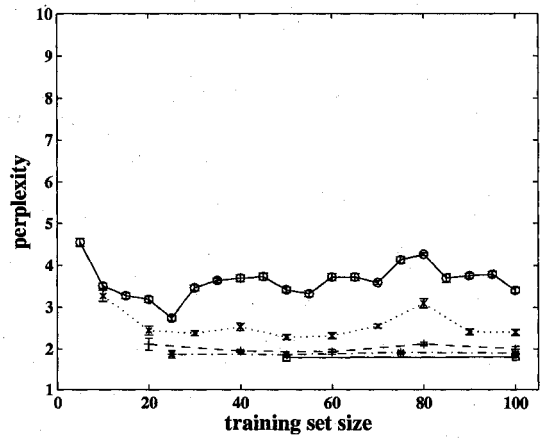
(b) igEM



(c) oigEM ($\chi = 0.25$)

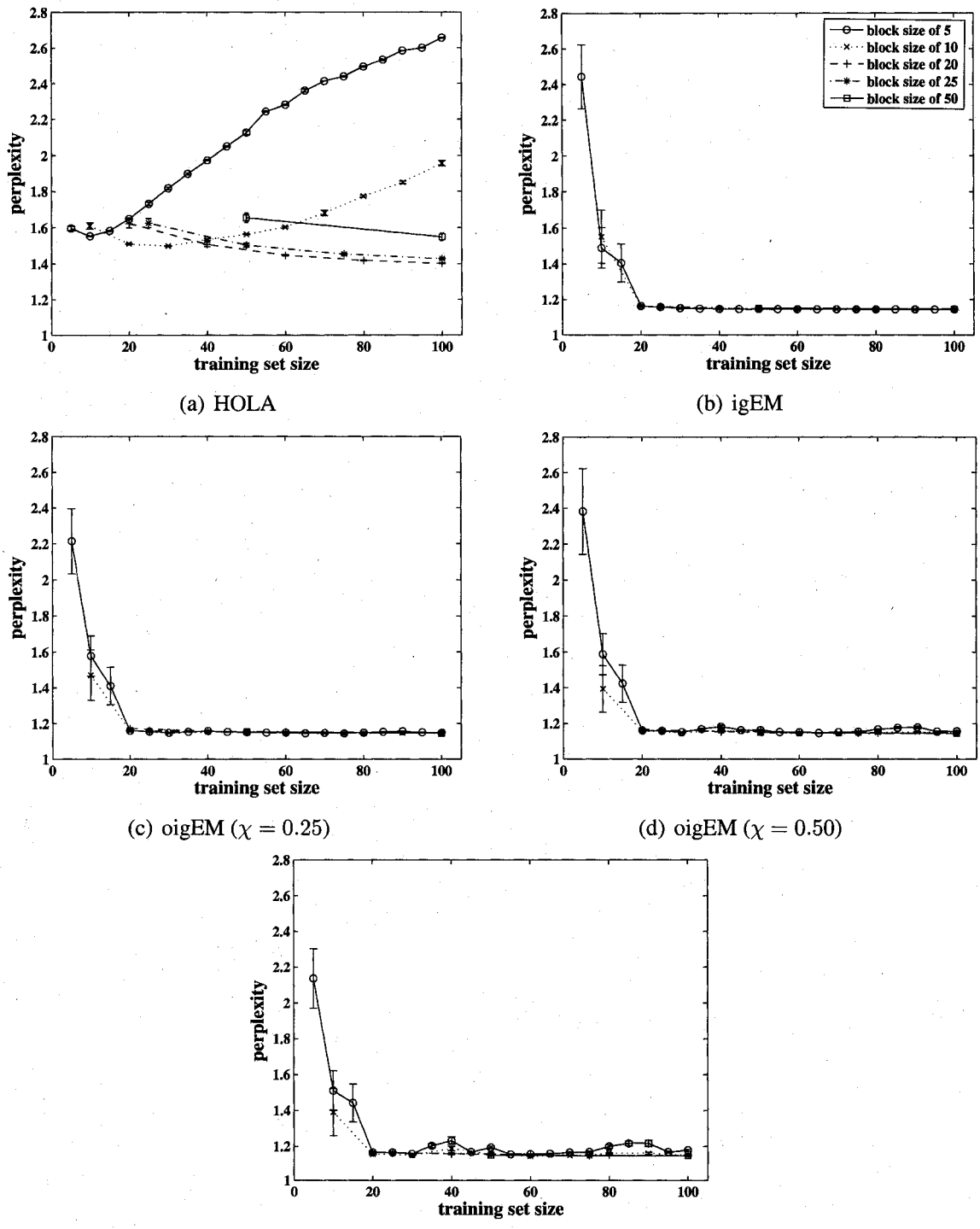


(d) oigEM ($\chi = 0.50$)



(e) oigEM ($\chi = 0.75$)

Figure 22 Average perplexity (PP) of a SCFG obtained by incremental learning with gEM/TS(IO), gEM/TS(VS) and HOLA on Mercury data for different block sizes. Error bars are standard error of the sample mean.



(a) HOLA

(b) igEM

(c) oigEM ($\chi = 0.25$)

(d) oigEM ($\chi = 0.50$)

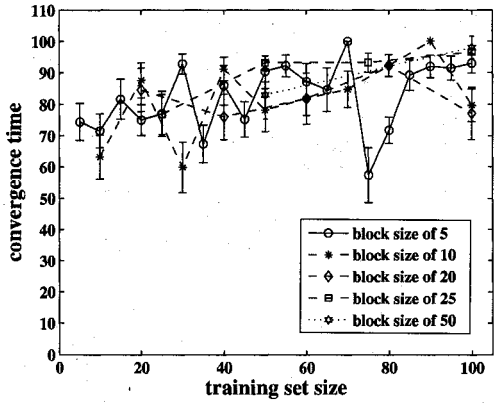
(e) oigEM ($\chi = 0.75$)

Figure 23 Average perplexity (PP) of a SCFG obtained by incremental learning with gEM/TS(IO), gEM/TS(VS) and HOLA on Pluto data for different block sizes. Error bars are standard error of the sample mean.

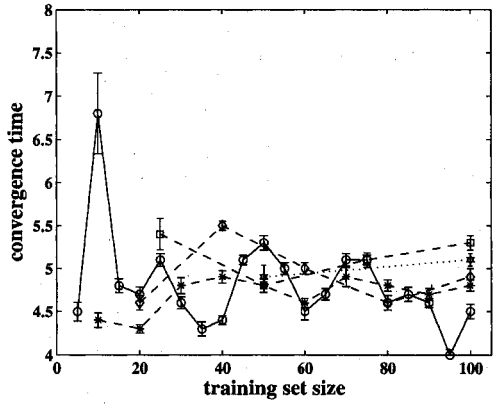
Therefore, the first point of each igEM and oigEM curve in Fig. 22 and 23 corresponds to the performance of gEM. In contrast, the perplexity of a SCFG obtained with HOLA never tends toward that of a SCFG trained though learning on a one single cumulative block. It can be observed that the standard error of the mean is very small for igEM and oigEM, whatever the value of χ. In contrast, for HOLA, it increases with the size of blocks.

The ambiguity of the Pluto grammar is greater, yet its complexity is lower. With Pluto data, the perplexity of a SCFG obtained by incremental learning with either igEM or oigEM also tends toward the behaviour of a SCFG obtained by batch learning with gEM. It appears that a $|\Omega_i| = 5$ sequences is sufficient to reach the lowest possible perplexity. At that point, incremental learning with igEM or oigEM gives the same result as batch learning with gEM. In contrast, HOLA requires at least a training block size of 20 sequences to converge. The fact that HOLA converges for Pluto and not for Mercury may be linked to the lower number of production rules with Pluto. Tuning the learning rate χ of oigEM does not have a significant impact on the overall perplexity when using training blocks larger than 10 sequences. For training block sizes of 5 and 10 sequences, it appears that the perplexity is less stable if χ is high. Indeed, increasing the value of χ assign a greater importance to the new data in the learning process. However, the overall perplexity keeps the same order of magnitude. It can be observed that, for experiments on Pluto data, the standard error of the mean that appears for the training on the first blocks of sequences for $|\Omega_i| = 5$ and 10 with igEM and oigEM becomes very small once the algorithm has converged to a minimum perplexity. The standard error of the mean with HOLA is always very small, whatever the size of blocks.

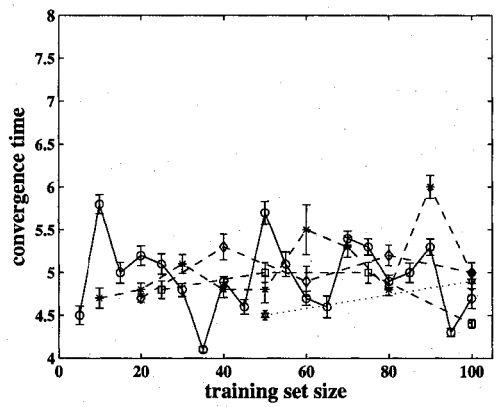
Figs. 24 and 25 display the average convergence time needed by igEM, oigEM and HOLA to learn Mercury and Pluto data, respectively, for different block sizes $|\Omega_i|$. This figure corresponds to the results found in Figs. 22 and 23. For each block size, the average convergence time on the Mercury data ranges from four to eight iterations across block sizes for igEM and oigEM, whereas it ranges from 50 to 100 iterations for HOLA. This is



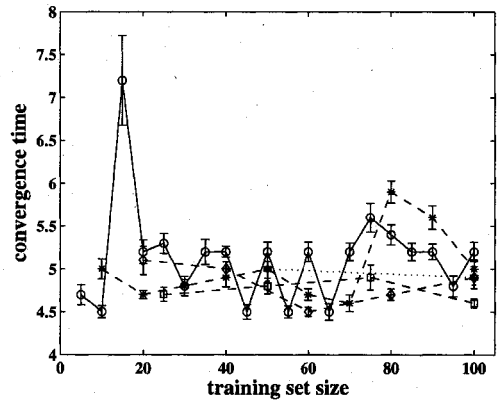
(a) HOLA



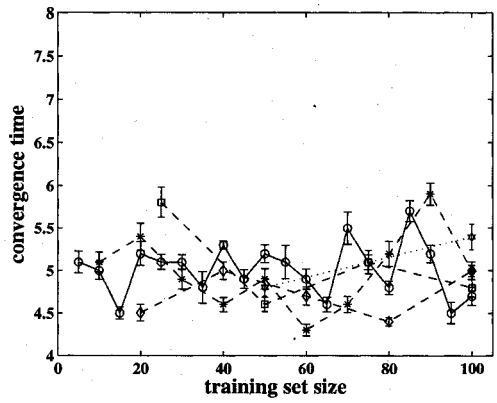
(b) igEM



(c) oigEM ($\chi = 0.25$)

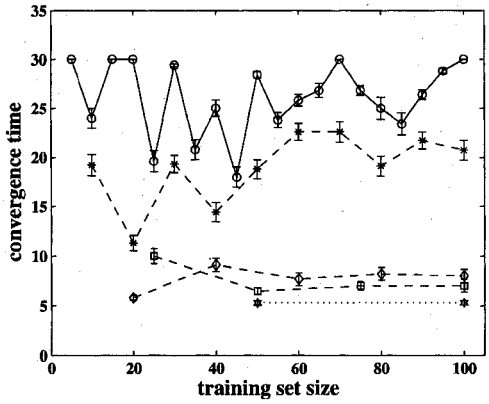


(d) oigEM ($\chi = 0.50$)

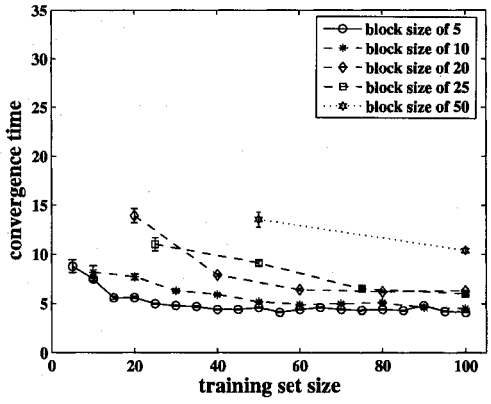


(e) oigEM ($\chi = 0.75$)

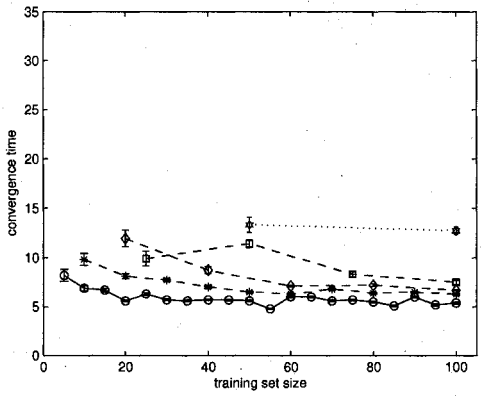
Figure 24 Average convergence time (I) of a SCFG obtained by incremental learning with igEM, oigEM and HOLA on Mercury data for different block sizes. Error bars are standard error of the sample mean.



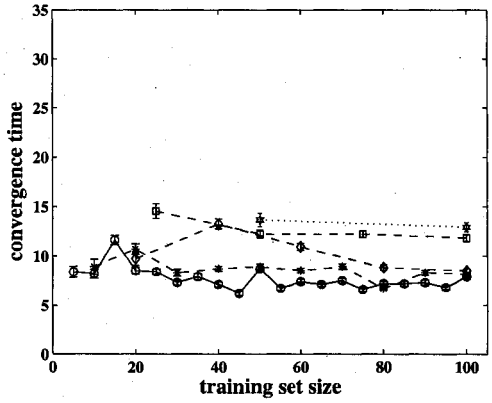
(a) HOLA



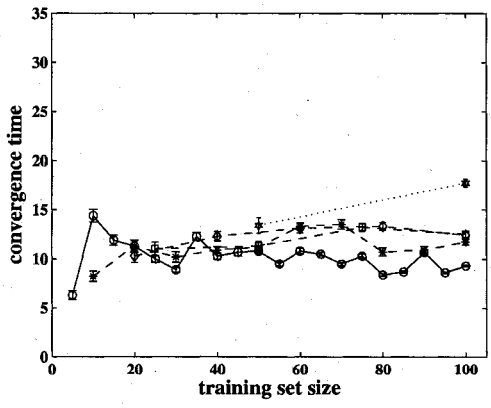
(b) igEM



(c) oigEM ($\chi = 0.25$)



(d) oigEM ($\chi = 0.50$)



(e) oigEM ($\chi = 0.75$)

Figure 25 Average convergence time (I) of a SCFG obtained by incremental learning with igEM, oigEM and HOLA on Pluto data for different block sizes. Error bars are standard error of the sample mean.

consistent with the behaviour of HOLA, since it has difficulty converging toward a minimum. The convergence time tends to fluctuate, but remains relatively constant when new blocks are learned. A peak can be observed for the second of the third block for igEM, oigEM($\chi = 0.25$), and oigEM($\chi = 0.50$), when $|\Omega_i| = 5$. This comes from the fact that the sequences of the blocks are quite different from one block to another. Peaks do not appear for larger blocks, since they share a greater number of similar sequences. It emerges from theses graphs that the size of training blocks does not seem to have significant influence on the convergence time for experiments on Mercury data. Finally, the standard error of the mean does not seem to be influenced by either the size of blocks or by the fact that new blocks are learned incremnetally.

Experiments on Pluto data gives different results. First, as with batch learning in Section 6.1, the convergence time is higher than for experiments on Mercury data, and ranges from four to sixteen iterations. When new blocks are added, the convergence time decreases for experiments with igEM, for all the block sizes. This can also be observed for oigEM($\chi = 0.25$). However, increasing the value of χ tends to reverse this behaviour. When new blocks are added, the convergence time remains almost constant for oigEM($\chi = 0.50$), and increases for oigEM($\chi = 0.75$). Moreover, the igEM and oigEM techniques have similar convergence times when $\chi = 0.25$, but it tends to increase for oigEM as χ grows beyond 0.25. This can be explained by the fact that increasing χ gives more importance on new data. The system therefore needs a greater number of iterations to adapt to the new data.

It can also be observed that increasing the length of the training blocks increases the corresponding standard error of the mean, which is normal, since training with more examples of a class gives more information on this class, and therefore makes its representation more complex. Different initializations of the probabilities will then act differently to converge toward an optimum. Increasing the size of blocks does not seems to modify the standard error of the mean for igEM and oigEM. In contrast, adding new blocks makes it decrease,

whatever the technique. Experiments with HOLA show that passing from $|\Omega_i| = 10$ to $|\Omega_i| = 20$ significantly decreases the convergence time. This is consistent with the results displayed in Fig. 23 (a), since it indicates that a minimum of 20 sequences per block is required in order to make HOLA converge. Then, as previously explained in Section 6.1, increasing the size of blocks beyond 20 sequences does not have a significant impact on the convergence time, since. Finally, once the size of blocks is large enough to make the algorithm converge, its convergence time have a low standard error of the mean.

Tables X and XI display the confusion matrices of the "best" SCFGs on the MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$ datasets, after training with $|\Omega_i|$=5 sequences. Recall that the "best" SCFG corresponds to the SCFG that obtain the lowest perplexity on Test. When comparing these confusion matrices to those obtained with batch learning, it can be seen that SCFGs obtained after learning with igEM and oigEM behave consistently, while the SCFG obtained after learning with HOLA produced a considerable number of classification errors for TN1 and TN2. Results indicate that HOLA does not support incremental learning as well as either igEM and oigEM, while results obtained with igEM and iogEM are very similar. Moreover, the SCFG obtained after learning with HOLA was not able to recognize states for two sequences of the datasets, due to the additional rules needed to to smooth the grammar.

In Table X, when MTI is processed by all algorithms, the estimation errors only occur when Na is estimated instead of Acq, while far more errors occur in Table V. This confirms our previous interpretation of ambiguity for MFRs (see Section 5.1), whereby classification errors can only appear when several parse trees exist for a same sequence. With MTI, ambiguity only emerges from the fact that Acq and Na can produce the same phrase. There is more tolerance in the noisy grammars used for $\texttt{TN}_1$ and $\texttt{TN}_2$.

Like with the batch learning experimetns, the ability of SCFGs to estimate radar states degrades with noisy data. Recall that the last rows and columns of the tables still represent

Table X

Confusion matrix associated with MTI, MTN_1, and MTN_2 for SCFGs obtained through incremental learning of MTrain with igEM, oigEM and HOLA, using $|\Omega_i| = 5$.

			Estimated States					Recognition rate per state
			S TI/TN_1/TN_2	**Acq** TI/TN_1/TN_2	**Na** TI/TN_1/TN_2	**Rr** TI/TN_1/TN_2	**Tm** TI/TN_1/TN_2	
Real States	**S**	igEM	4178 / 3174 / 2381	0 / 110 / 347	0 / 138 /161	0 / 33 / 119	0 / 103 / 416	**1 / 0.892 / 0.695**
		oigEM	4178 / 3641 / 2273	0 / 121 / 360	0 / 114 / 222	0 / 18 / 142	0 / 101 / 422	**1 / 0.911 / 0.665**
		HOLA	4178 /1043 / 840	0 / 473 / 522	0 / 159 / 188	0 / 133 / 202	0 / 153 / 398	**1 / 0.532 / 0.391**
	Acq	igEM	0 / 10 / 32	1734 / 1511 / 948	235 / 333 / 350	0 / 9 / 60	0 / 22 / 76	**0.88 / 0.802 / 0.647**
		oigEM	0 / 8 / 45	1734 / 1516 / 926	235 / 332 / 364	0 / 10 / 54	0 / 13 / 79	**0.88 / 0.807 / 0.631**
		HOLA	0 / 0 / 23	1734 / 986 / 467	235 / 129 / 158	0 / 89 / 110	0 / 2 / 111	**0.88 / 0.818 / 0.537**
	Na	igEM	0 / 0 / 0	0 / 7 / 10	736 / 599 / 388	0 / 92 / 199	0 / 0 / 39	**1 / 0.858 / 0.610**
		oigEM	0 / 0 / 0	0 / 8 / 27	736 / 608 / 383	0 / 82 / 193	0 / 0 / 40	**1 / 0.871 / 0.596**
		HOLA	0 / 0 / 2	0 / 10 / 7	736 / 399 / 335	0 / 158 / 99	0 / 0 / 21	**1 / 0.704 / 0.722**
	Rr	igEM	0 / 31 / 34	0 / 0 / 32	0 / 79 / 167	2073 / 1802 / 1150	0 / 66 / 373	**1 / 0.911 / 0.655**
		oigEM	0 / 31 / 39	0 / 0 / 29	0 / 51 / 182	2073 / 1845 / 1133	0 / 56 / 358	**1 / 0.93 / 0.650**
		HOLA	0 / 2 / 49	0 / 3 / 47	0 / 373 / 422	2073 / 1129 / 592	0 / 168 / 354	**1 / 0.674 / 0.404**
	Tm	igEM	0 / 759 / 1099	0 / 62 / 192	0 / 280 / 427	0 / 75 / 274	6243 / 5111 / 4272	**1 / 0.813 / 0.682**
		oigEM	0 / 485 / 1265	0 / 61 / 193	0 / 380 / 465	0 / 74 / 246	6243 / 5340 / 4094	**1 / 0.842 / 0.654**
		HOLA	0 / 2814 / 2342	0 / 149 / 363	0 / 690 / 852	0 / 200 / 580	6243 / 1915 / 1691	**1 / 0.332 / 0.290**
Confidence on estimated state		**igEM**	**1 / 0.799 / 0.671**	**1 / 0.894 / 0.620**	**0.758 / 0.419 / 0.26**	**1 / 0.896 / 0.676**	**1 / 0.964 / 0.825**	
		oigEM	**1 / 0.874 / 0.628**	**1 / 0.888 / 0.603**	**0.758 / 0.409 / 0.237**	**1 / 0.909 / 0.539**	**1 / 0.969 / 0.82**	
		HOLA	**1 / 0.270 / 0.258**	**1 / 0.608 / 0.332**	**0.758 / 0.228 / 0.171**	**1 / 0.661 / 0.374**	**1 / 0.856 / 0.657**	

Table XI

Confusion matrix associated with PTI, PTN_1, and PTN_2 for SCFGs obtained through incremental learning of PTrain with igEM, oigEM and HOLA, using $|\Omega_i| = 5$.

Real States			Estimated States: S (TI/TN_1/TN_2)	Estimated States: Na (TI/TN_1/TN_2)	Estimated States: Rr (TI/TN_1/TN_2)	Estimated States: Tm (TI/TN_1/TN_2)	Recognition rate per state
S	S	igEM	9041 / 12912 / 9856	5517 / 2728 / 1668	0 / 9 / 1042	1582 / 124 / 1671	**0.56 / 0.819 / 0.692**
		oigEM	9041 / 13034 / 9726	5517 / 2561 / 1663	0 / 9 / 1098	1582 / 151 / 1689	**0.56 / 0.827 / 0.686**
		HOLA	183 / 7 / 178	14375 / 14391 / 8642	0 / 579 / 2168	1582 / 435 / 2857	**0.011 / 0.0003 / 0.013**
Acq	Acq	igEM	0 / 326 / 325	1845 / 1151 / 263	0 / 314 / 809	0 / 0 / 327	**1 / 0.643 / 0.156**
		oigEM	0 / 322 / 323	1845 / 1141 / 251	0 / 324 / 784	0 / 0 / 356	**1 / 0.639 / 0.146**
		HOLA	0 / 4 / 30	1845 / 1102 / 260	0 / 563 / 605	0 / 86 / 812	**1 / 0.628 / 0.152**
Na	Na	igEM	0 / 66 / 306	0 / 0 / 29	2016 / 1647 / 431	0 / 266 / 1061	**1 / 0.832 / 0.236**
		oigEM	0 / 66 / 306	0 / 0 / 24	2016 / 1637 / 416	0 / 274 / 1073	**1 / 0.828 / 0.229**
		HOLA	0 / 25 / 54	0 / 104 / 392	2016 / 1242 / 421	0 / 591 / 932	**1 / 0.633 / 0.234**
Tm	Tm	igEM	2685 / 3245 / 4049	0 / 142 / 486	0 / 0 / 421	8928 / 8026 / 5535	**0.769 / 0.703 / 0.528**
		oigEM	2685 / 3250 / 4053	0 / 142 / 486	0 / 0 / 405	8928 / 8017 / 5520	**0.769 / 0.703 / 0.543**
		HOLA	17 / 7 / 197	2668 / 3896 / 4899	0 / 87 / 1400	8928 / 7171 / 3757	**0.769 / 0.643 / 0.366**
Confidence on estimated state		**igEM**	**0.771 / 0.780 / 0.678**	**0.749 / 0.286 / 0.107**	**1 / 0.836 / 0.157**	**0.85 / 0.954 / 0.648**	
		oigEM	**0.771 / 0.782 / 0.675**	**0.749 / 0.297 / 0.104**	**1 / 0.831 / 0.154**	**0.85 / 0.950 / 0.649**	
		HOLA	**0.915 / 0.096 / 0.388**	**0.098 / 0.057 / 0.018**	**1 / 0.503 / 0.092**	**0.85 / 0.912 / 0.45**	

the classification rate per state, from two different points of view. The last row indicates the confidence that can be associated with an estimated state – that is the ratio of corrected estimated state on the total number of times this states was estimated –, while the last column indicates the ability to recognize the real states – that is the ratio of corrected estimated state on the total number of times this states really appears.

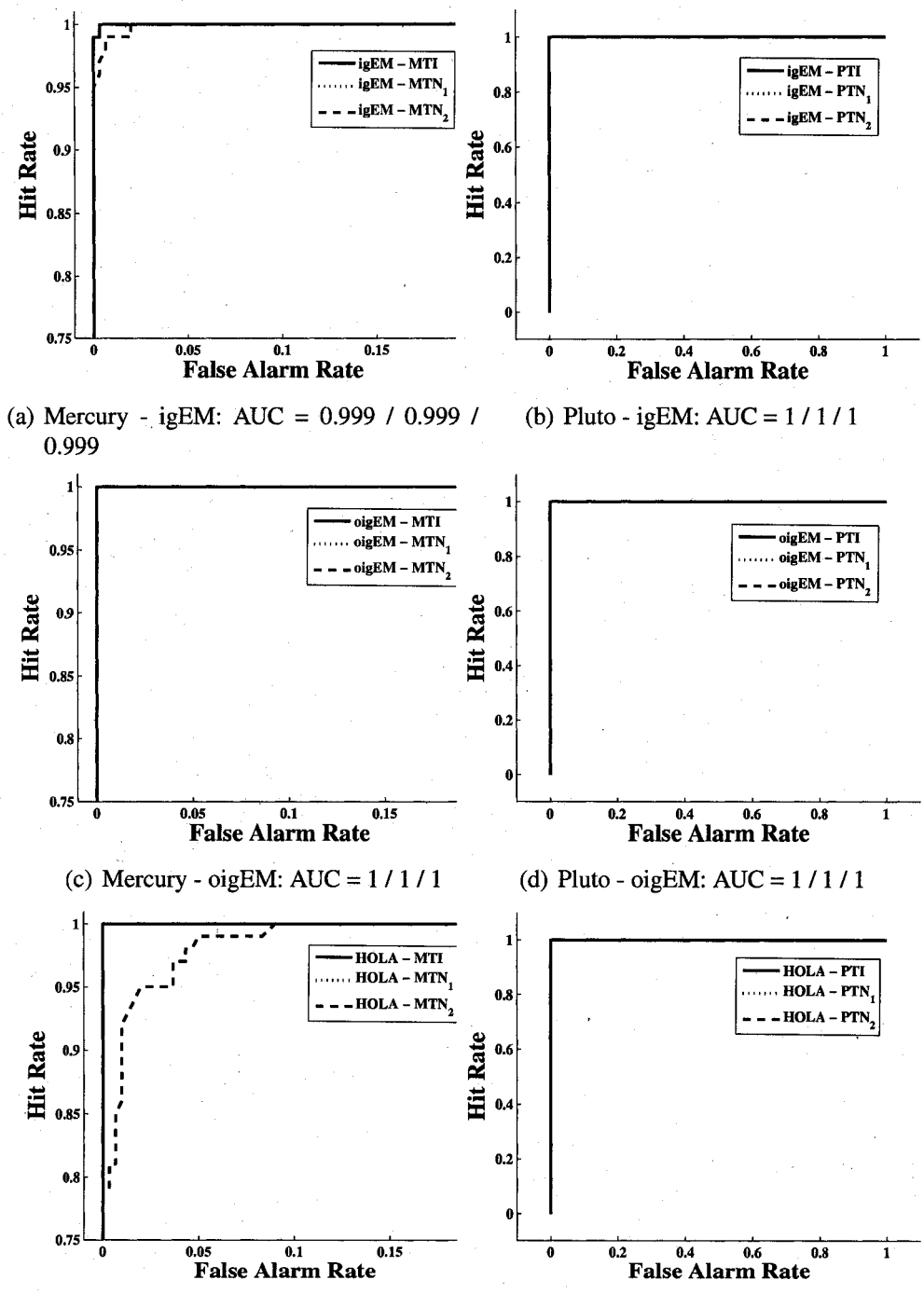
Table XII

Perplexity of SCFGs obtained with igEM, oigEM, and HOLA on MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$.

	MERCURY			PLUTO		
Test subset	**igEM**	**oigEM**	**HOLA**	**igEM**	**oigEM**	**HOLA**
MTI/PTI	2.85	2.51	3.34	1.14	1.28	2.56
$\texttt{MTN}_1/\texttt{PTN}_1$	4.09	3.96	6.93	1.76	1.76	1.60
$\texttt{MTN}_2/\texttt{PTN}_2$	9.99	8.72	10.39	4.49	4.49	4.49

Table XII shows the mean perplexity obtained for the "best" SCFG on the MTI/PTI, $\texttt{MTN}_1/\texttt{PTN}_1$, and $\texttt{MTN}_2/\texttt{PTN}_2$ test sets. Although perplexity tends to grow with the level of noise for all three techniques, as with the experiments on MTest/PTest, the perplexity obtained with igEM and oigEM is always significantly lower than with HOLA. All the algorithms give the same results for $\texttt{PTN}_2$ (the computation has reached the limit of precision for the computer).

Fig. 21 shows the ROC curves for the Mercury and Pluto grammars, using the "best" SCFGs on MROCTI, $\texttt{MROCTN}_1$, $\texttt{MROCTN}_2$, PROCTI, $\texttt{PROCTN}_1$, and $\texttt{PROCTN}_2$ databases. Recall that these databases are composed of 400 sequences. The first 100 sequences of MROCTI, $\texttt{MROCTN}_1$, and $\texttt{MROCTN}_2$ belong to Mercury language, whereas the last 300 sequences belong to the three other MFRs languages. The first 100 sequences of PROCTI, $\texttt{PROCTN}_1$, and $\texttt{PROCTN}_2$ belong to Pluto language, whereas the last 300 sequences belong to the three other MFRs languages. By computing the perplexity associated with each test



(a) Mercury - igEM: AUC = 0.999 / 0.999 / 0.999

(b) Pluto - igEM: AUC = 1 / 1 / 1

(c) Mercury - oigEM: AUC = 1 / 1 / 1

(d) Pluto - oigEM: AUC = 1 / 1 / 1

(e) Mercury - HOLA: AUC = 1 / 1 / 0.996

(f) Pluto - HOLA: AUC = 1 / 1 / 1

Figure 26 Average perplexity for HOLA, igEM and oigEM versus `PTrain` size, for different block sizes. (Error bars are standard error of the sample mean).

sequence and by varying a threshold across the range of perplexity values on can compute the Hit Rate and the False Alarm Rate of a SCFG (see Section 5.3.4).

It can be seen that performance degrades when noise is introduced for the Mercury grammar, but that the results remains very discriminant even for a noise frequency of one error in 10 words. It even remains completely discriminant for oigEM. Here, incremental learning gives results that are comparable to those of batch learning. This may come from the fact that even if the perplexity of the sequences belonging to the corresponding MFR's language is higher, it is even higher for the sequences of the other MFRs. As for the ROC curves from batch learning, the analysis of the perplexity obtained on the whole databases MROCTI, $\texttt{MROCTN}_1$, and $\texttt{MROCTN}_2$ shows that the errors come from sequences from VenusA and VenusB, because their language appears to be quite close to Mercury's, while Pluto's language is more distinct from the three others. Indeed, these curves show that increasing noise to a frequency of one in 10 words for the replacements does not have a significant impact on the performance obtained with Pluto SCFGs, using either igEM, oigEM, or HOLA.

Overall, it appears that the SCFGs can always detect the MFR of interest with very high level of accuracy. The worst result is a ratio of (HR=0.96)/(FAR=0.04), given by HOLA on $\texttt{MROCTN}_2$, which corresponds to an extreme case, in which only the sequences emitted by the MFR of interest are noisy, with a high level of noise. Moreover, all the AUC remain higher than 0.9.

Table XIII displays the threshold γ^* that corresponds to the best discrimination, along with the corresponding values of HR and FAR, when the difference between HR and FAR is maximized: $\gamma^* = \text{argmax}\{HR(\gamma) - FAR(\gamma)\}$ (note that these operation point do not count for the costs of errors). For results of experiments on Pluto data, the threshold are quite similar to those obtained after using batch learning. As expected they are a little higher, which is caused by the perplexity of a few sequences. This may be explained by

the fact that the sequences learned at the beginning of incremental learning process have less influence than when batch learning is used. The thresholds obtained when incremental learning was used for training on Mercury data are significantly higher than when batch learning is used for training. This may signify that the grammar and especially language complexities, that are greater for Mercury than for Pluto, may have an impact on incremental learning. Indeed, for complex languages, the influence of the sequences learned at the beginning of incremental learning process would be even lower than for simple languages, for which these sequences are more likely to appear in the recent blocks of data.

Table XIII

Best threshold, and associated HR and FAR, for SCFGs obtained with gEM/TS(IO), gEM/TS(VS) and HOLA on MTI/PTI, MTN_1/PTN_1, and MTN_2/PTN_2.

	MERCURY		
	igEM MTI/MTN_1/MTN_2	**oigEM** MTI/MTN_1/MTN_2	**HOLA** MTI/MTN_1/MTN_2
γ^*	15.97 / 15.97 / 17.70	8.48 / 9.37 / 17.86	8.33 / 9.72 / 18.29
HR	1 / 1 / 0.99	1 / 1 / 1	1 / 1 / 0.97
FAR	0 / 0 /0	0 / 0 / 0	0 / 0 / 0.37

	PLUTO		
	igEM MTI/MTN_1/MTN_2	**oigEM** MTI/MTN_1/MTN_2	**HOLA** MTI/MTN_1/MTN_2
γ^*	5.12/6.44/9.83	5.66/6.78/10.86	11.35/14.03/21.64
HR	1/1/1	1/1/1	1/1/1
FAR	0/0/0	0/0/0	0/0/0

The importance, for the igEM and the oigEM techniques, of re-initializing the production rule probabilities with each new block of data was theoretically discussed in Section 4.3. Fig. 27 shows the evolution of the perplexity obtained with igEM versus the training sizes, after training on Mercury data when $|\Omega_i| = 1$ sequence. Incremental learning of blocks is performed with and without re-initializing the probabilities prior to learning each new block. The figure clearly shows that the perplexity is significantly higher if the production rule probabilities are not re-initialized. Learning using $|\Omega_i| = 1$ logically results in a very

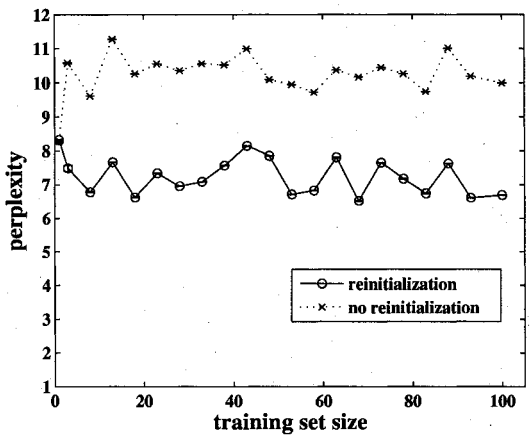


Figure 27 Average perplexity obtained with igEM versus the training sizes, after training on Mercury data with $|\Omega_i| = 1$ sequence. Incremental learning of blocks is performed with and without re-initializing the probabilities prior to learning each new block of training sequences.

low standard error of the mean, since the corresponding cost function to optimize is very simple each time. Since reinitializing the probabilities provides lower perplexity, it shows that igEM may be more likely to avoid local minima. Results therefore demonstrate the validity of the theoretical discussion of Section 4.3.

6.2.2 Resources requirements

Tables XIV and XV present the estimates of time complexity per iteration and memory complexity of the learning techniques used for incremental training. These tables use the same notations as in Section 6.1.2. For one sequence, time complexity per iteration and memory complexity only depend on the computational process of `Get-Inside-Probs` and of `Get-Expectation`. These two algorithms are left unchanged during the incremental learning of new data. Since the difference between gEM, igEM and oigEM therefore lies only in the re-estimation equation, time complexity per iteration and memory complexity are the same for the three algorithms. Therefore, the numerical values of T and M are the same as those displayed in Section 6.1.2.

Table XIV

Estimates of time complexity per iteration of learning techniques.

	T		
Techniques	**Average**	**Worst-case**	**Growth rate**
gEM, igEM and oigEM	$6\,\lvert\varphi_e\rvert + 9 \cdot \lvert\varphi_t\rvert \cdot \lvert\Delta l_t\rvert$	$6 \cdot M_{nt} \cdot L +$ $9 \cdot M_{nt}^3 \cdot L \cdot (L^2 - 1)$	$O(M_{nt}^3 \cdot L^3)$
HOLA	$2 \cdot \lvert r\rvert$	$2 \cdot (M_{nt}^3 + M_{nt} \cdot M_t)$	$O(M_{nt}^3)$

Table XV

Estimates of memory complexity of learning techniques.

	M		
Techniques	**Average**	**Worst-case**	**Growth rate**
gEM, igEM and oigEM	$L^2 \cdot M_{nt} \cdot (2 + \lvert\Delta_l\rvert) +$ $4 \cdot \lvert\varphi_e\rvert + 9 \cdot \lvert\varphi_t\rvert \cdot \lvert\Delta l_t\rvert$	$2 \cdot M_{nt} \cdot L \cdot (2 + L)$ $+ M_{nt}^3 \cdot (10 \cdot L^3 - 8)$	$O(M_{nt}^3 \cdot L^3)$
HOLA	$4 \cdot \lvert r\rvert$	$4 \cdot (M_{nt}^3 + M_t \cdot M_{nt})$	$O(M_{nt}^3)$

An overall measure of the complexity needed to learn a new block of training sequences would however reveal the true impact on performance of incremental learning. Recall that the overall time and memory complexities needed to learn a block Ω_i are $Tj = T_{\text{init}} + T \cdot \Gamma_T \cdot I$ and $M^* = M \cdot \Gamma_M$, where Γ_T and Γ_M are multiplicative factors that depend on the size of $|\Omega_i|$, I is the convergence time, and T and M are the time complexity per iteration and the memory complexity. Consider that incremental learning algorithms are trained on a dataset Ω, divided into n blocks $\Omega_{i=1,\ldots,n}$, and that each new block is available once training was completed on the previous ones. The factors for incremental learning techniques, Γ_T and Γ_M, are presented in Table XVI. For gEM, Γ_T corresponds to the sum of the number of sequences for all the successive databases, and Γ_M corresponds to the current number of sequences to store. For igEM and oigEM, Γ_T and Γ_M correspond to the current number of sequences of the block. For HOLA, the values of Γ_T and Γ_M

corresponds to the number of sequences it generates at each iteration. Results indicate that incremental learning provides a considerable saving in terms of computational resources. In addition HOLA provides the lowest overall time and memory complexities.

Table XVI

Multiplicative factors for the computation of the overall time and space complexities associated with gEM, igEM and iogEM techniques used for incremental learning. n is the total number of blocks.

	gEM	igEM / oigEM	HOLA
Γ_T	$\sum_{j=1}^{n}\sum_{i=1}^{j}\lvert\Omega_i\rvert$	$\sum_{i=1}^{n}\lvert\Omega_i\rvert$	number of generated sequences
Γ_M	$\sum_{i=1}^{n}\lvert\Omega_i\rvert$	$\lvert\Omega_n\rvert$	1

It is now possible to compute the overall time T^* and memory M^* complexity for gEM, igEM, oigEM, and HOLA. The influence of the preprocessing T_{init} is not considered. Table XVII gives the numerical values of Γ_T and Γ_M for gEM, igEM, oigEM and HOLA, when considering that incremental learning has been performed on a database of 100 sequences, with $|\Omega_i| = 5$. Let's consider that HOLA also generates 5 sequences each time.

Table XVII

Numerical values of the multiplicative factors for the computation of the overall time and space complexities associated with gEM, igEM and iogEM techniques used for incremental learning.

	gEM	igEM / oigEM	HOLA
Γ_T	1050	100	5
Γ_M	100	5	1

Since M^* is independent from the convergence time, it has the same value for igEM and oigEM. Therefore, for the experiments on Mercury data, T^* and M^* are:

- $T^*_{HOLA} = 7.05 \cdot 10^4 < T^*_{igEM} = 4.69 \cdot 10^6 < T^*_{oigEM(\chi=0.25)} = 4.79 \cdot 10^6 < T^*_{oigEM(\chi=0.75)} = 4.89 \cdot 10^6 < T^*_{oigEM(\chi=0.50)} = 4.99 \cdot 10^6 < T^*_{gEM} = 5.13 \cdot 10^7$;
- $M^*_{HOLA} = 14.08 \cdot 10^3 < M^*_{igEM/oigEM} = 2.18 \cdot 10^8 < M^*_{gEM} = 4.36 \cdot 10^9$.

For the experiments on Pluto data, T^* and M^* are:

- $T^*_{HOLA} = 3.15 \cdot 10^4 < T^*_{igEM} = 9.40 \cdot 10^6 < T^*_{oigEM(\chi=0.25)} = 11.28 \cdot 10^6 < T^*_{oigEM(\chi=0.50)} = 14.10 \cdot 10^6 < T^*_{oigEM(\chi=0.75)} = 18.80 \cdot 10^6 < T^*_{gEM} = 98.70 \cdot 10^6$;
- $M^*_{HOLA} = 7.00 \cdot 10^3 < M^*_{igEM/oigEM} = 1.12 \cdot 10^8 < M^*_{gEM} = 2.24 \cdot 10^9$.

Experiments show that HOLA still have the lower overall time T^* and memory M^* complexities. A great difference appears here between gEM and its incremental versions, igEM and oigEM. This proves that these two techniques, that allow for incremental learning, can significantly faster the re-estimation of probabilities. The influence of χ on T^* is of course the same as its influence on I. Varying the value of parameter χ does not have a considerable impact on T^*. Since M^* is independent from I, χ does not affect its numerical values.

CHAPTER 7

CONCLUSION

Radar Electronic Support (ES) systems are employed in the context of military surveillance, to search for, intercept, locate, analyze and identify radiated electromagnetic energy. Two critical functions of these systems are the recognition of radar emitters associated with intercepted pulse trains, and the estimation of the instantaneous level of threat posed by these radars. In modern environments, the recent proliferation and complexity of electromagnetic signals encountered by radar ES systems is greatly complicating these functions. In order to exploit the dynamic nature of many modern radar systems, advanced signal processing algorithms based on Stochastic Context Free Grammars (SCFGs) have been proposed for modeling the behavior of multi-function radar (MFR) systems.

A challenge to the practical application of SCFGs is the task of learning probability distributions associated with the production rules of the grammars. The most popular techniques for this task are the Inside-Outside (IO) and Viterbi Score (VS) algorithms. Unfortunately, the application of IO and VS techniques to real-world ES problems is limited due to the time and memory complexity per iteration and to the large number of iterations needed for learning. Moreover, since new information from a battlefield or other sources often becomes available at different points in time, fast incremental learning of SCFG probabilities is important for rapidly reflecting changes in operational environments. The objective of this thesis is to explore and develop fast incremental techniques that are suitable for learning SCFG probabilities in radar ES applications.

In the first part of this thesis, three fast alternatives to IO and VS, called Tree Scanning (TS), graphical EM (gEM), and HOLA, are first compared from several perspectives. Unlike traditional implementations of IO and VS, these techniques rely on the precomputation of data structures to accelerate the probability re-estimation process. Since

VS may have a lower computational cost in practice, VS versions of the original IO-based gEM and TS have also been proposed and compared. These variants are denoted by gEM(VS) and TS(VS) to distinguish from the original algorithms, denoted by gEM(IO) and TS(IO). The IO and VS versions of TS and gEM are mathematically equivalent to the classical IO and VS algorithms respectively, while numerically, they may differ due to finite machine precision. An experimental protocol has been defined to assess impact on performance of different training set sizes, sequences lengths, and levels of grammar ambiguity, etc. Proof-of-concept computer simulations have been performed on using synthetic radar pulse data from different types of MFR systems. The performance of these techniques has been measured in terms of resource allocation and accuracy.

Results indicate that gEM(IO), gEM(VS), TS(IO) and TS(VS) systematically provide a higher level of accuracy than HOLA, yet have significantly higher time and memory complexities. Unless the MFR system is modeled by a very ambiguous SCFG, these techniques can learn probabilities rapidly, at the expense of significantly higher memory resources. Results have revealed that TS appears to be more efficient for less ambiguous grammars, such as with the Mercury MFR, since it enumerates all possible trees. In contrast gEM becomes a better choice when the level of ambiguity increases, as with the Pluto MFR, since the parse trees are compactly represented by support graphs. VS versions of gEM and TS do not degrade accuracy with respect to the IO versions, yet provide a significantly lower convergence time and time complexity per iteration in practice. Finally, the execution time and memory requirements of HOLA are orders of magnitude lower than that of gEM and TS. The computational complexity of HOLA is bound by the number of SCFG rules, not by the amount of training data. Furthermore, HOLA is an on-line technique that can update these probabilities incrementally on the fly, in order to reflect changes in the environment. With the exception of HOLA, the iterative re-estimation process has to be started from the beginning to account for new training data.

In the second part of this thesis, two new incremental derivations of the original gEM algorithm, called incremental gEM (igEM) and on-line incremental gEM (oigEM), are proposed and compared to HOLA. As with the original gEM, they rely on the pre-computation of data structures to accelerate the probability re-estimation process. In addition, they allow to learn new training data incrementally, without accumulating and storing all training sequences, and without re-training a SCFG from the start using all cumulative data. A second experimental protocol has been defined to assess impact on performance of factors such as the size of the successive training blocks, and levels of grammar ambiguity. Proof-of-concept computer simulations have been performed on using synthetic radar pulse data from different types of MFR systems. The performance of these techniques has been measured in terms of resource allocation and accuracy.

Results indicate that incremental learning of data bocks with igEM and oigEM provides the same level of accuracy as learning from all cumulative data from scratch, even for small data blocks. However, for complex grammars, a minimum block size is required. As expected, incremental learning significantly reduces the overall time and memory complexities. The igEM and oigEM algorithms systematically provide a higher level of accuracy than HOLA, especially for small block of data, in terms of perplexity and estimation of the state. All the algorithms behave consistently when considering MFRs recognition. Unless the MFR system is modeled by a very ambiguous SCFG, these techniques can learn probabilities rapidly, at the expense of significantly higher memory resources. The execution time and memory requirements of HOLA are orders of magnitude lower than that of igEM and oigEM.

CHAPTER 8

RECOMMENDATIONS

The research in this thesis contributes to a larger effort that seeks to assess the potential of SCFGs for detecting different types of MFRs, and for estimating their state at a given time, in radar ES applications. Based on function description of a MFR, the structure of a SCFG may be designed to model its dynamic behaviour. The results presented in the first part of this thesis provide guidance in the selection of fast techniques for learning the production rules of SCFGs in radar ES. It contains a comprehensive comparison of the performance achieved as a result of using different fast alternatives to IO and VS on synthetic radar data, namely TS, gEM and HOLA. Therefore, the selection of either technique would ultimately depend on the specifics of the ES application, and effect a trade-off between accuracy and computational efficiency. For instance, the HOLA technique appears more suitable for deployment in compact radar ES systems that require timely protection against threats. In contrast, the gEM and TS techniques could better address the requirements of radar ES systems for long range surveillance, where a slower response time may be tolerated for enhanced precision.

The quality of results and the resources requirements are important in radar ES applications. Since new training data is often available in blocks, incremental learning techniques, that allow for fast updates of SCFG probabilities, can be an undisputed asset. In light of the results presented in the first part of this thesis, two options were considered – either make HOLA more accurate, or make gEM and TS incremental. In the second part of this thesis , and two incremental versions of gEM, named igEM and oigEM, were developed, and compared to HOLA (note that the igEM and oigEM techniques easily adapt to TS). In this context, it appears that, if a MFR is modeled by a complex grammar, HOLA is not always suitable for incremental learning. For MFRs modeled by less complex grammars, HOLA requires a minimum number of new sequences per block in order to converge. Otherwise,

its performance in terms of perplexity and state estimation becomes an issue. Therefore, one may better consider either the igEM or oigEM technique. They both give good results when incremental learning is performed, and allow accelerate learning with respect to the classical gEM. The oigEM technique has the advantage of being parametrized, giving more or less influence to new data. For example, this parameter may be set according to a confidence associated with the condition of acquisition of new data blocks.

A wide range of techniques linked to the learning SCFG probabilities were studied during the course of this research. Several aspects deserved a more detailed investigation. The rest of this chapter outlines some suggestions for future research.

For instance, a straightforward extension of our study on incremental techniques would be to derive VS versions of igEM and oigEM. This would involve replacing the Get-Inside-Probs() and Get-Expectations() routines by Get-Inside-Probs-VS() and Get-Expectations-VS() routines in order to compute $\hat{\eta}$ on the best parse trees instead of η on all the parse tees of the training sequences. Indeed, despite the fact that no difference could be observed for experiments on Mercury data, it has been shown that the VS version of gEM allows accelerating convergence time for experiments on Pluto data.

Due to some time limitations, results obtained after training with incremental techniques were only assessed once the last block of data had been learned. However, it does not necessarily correspond to the best possible result, when considering the detection ability and the classification rate over radar states. If one wants to train a SCFG using the incremental techniques presented here, it would be suitable to test the results after training on each block of data, and assess the impact of the SCFG's perplexity on these measures, in order to select the set of production rule probabilities giving the best results.

HMMs have previously been considered for recognizing MFRs and for estimating their instantaneous level of threat. The complexity of algorithms needed to exploit HMMs are lower than those of SCFGs, but they lack the ability to efficiently model the behavior of

MFRs. Dynamic Bayesian Networks (DBNs) represent a third alternative – they would provide a more powerful method that HMMs for modeling MFR behavior. Furthermore, they allow for dependencies between observed and hidden states that are not allowed by SCFGs. This last propoerty may be useful for taking into account the dependencies between the words of consecutive MFR phrases, (which is not possible with SCFGs). Moreover, a MFR can follow several targets at a time using different tracks. The influence of state of a track on the other ones was not considered here, because it would have required very complex grammars, therefore very long to train and requiring very high resources requirements. DBN adaptability may allow modeling more efficiently this aspect of MFR functioning. However, this remains a hypothesis, and a study of learning techniques for DBN should be lead in order to determine if DBNs can be used in radar ES applications.

In real-world ES applications, long sequences of radar words may necessitate fast parsing and analysis techniques. Ultimately, performing on-line parsing would be an undisputed enhancement to the techniques studied in this thesis. For instance, Kato *et al.* (2005) have introduced an incremental dependency parsing method, which incrementally computes all dependency structures that are grammatically probable in a sequence. According to the authors, this method increases parsing speed ten fold.

Several techniques exist for smoothing of SCFGs (Jauvin, 2003), in order to be able to handle noise in the sequences. A basic method was exploited in this thesis. It allows to handle with any level of noise, without drastically increasing the number of possible derivation trees. A more detailed investigation of smoothing methods may have resulted in better results in tests that involve noisy data. Moreover, hypothesis was set in order to model noise at a word level. It consists in considering that noise can only appear as word replacement, without considering missing or additional words. Modeling these kinds of noise would require a very complex grammar, that would probably be unadapted to radar ES application, due to the associated resulting computational time and resources requirements. Although this hypothesis seems reasonable, its main weakness is that it does

not consider the fact that noise originally appears in the pulse sequences. Some function could have been integrated in the tokenizer in order to suppress or at least to decrease that noise. On-line parsing of the word sequences resulting from the tokenizer may also allow for detecting the erroneous words as soon as they appear, therefore allowing for more accurate state estimation. In this context, the overall quality of results from the radar ES system presented in Fig. 4 should be measured from results of the tokenizer and sequence recognition module.

In the experiments presented in this thesis, the training datasets were supposed totally noiseless, due to the fact that, in a real radar ES application, an analyst would scan the sequences before presenting them to the system. This allowed to train simpler SCFGs, with a well-defined structure, so that the states of the corresponding MFR could be easily identified in the derivation trees. However, it training smoothed grammars on noisy data could be a good option, and may give interesting results when testing on noisy data.

Regardless of the learning technique, stochastic grammars always require a considerable execution time when used for real-world applications. The main purpose of this work was to explore fast incremental techniques for learning SCFG probabilities in radar ES applications. One of the main weaknesses of this thesis is that computer simulations were mostly performed on synthetic data. Future work should include comparing the performance of these technique on complex real-world radar data. In order to assess the performance of gEM(IO) and gEM(VS) on different real-world data, some experiments were led on DNA data (see Appendix 5). DNA sequences are far less predictable than radar sequences, and require more ambiguous grammars. They also only have a four word vocabulary. These experiments allowed to confirm some differences between the learning techniques, such as a gap in the perplexity and approximate perplexity, and in the convergence time. However, it did not show significant difference in the task of species recognition. Finally, the task of DNA structure estimation (that corresponds to the ability of estimating the MFR states in radar ES applications), was not assessed during these experiments.

Finally, all the techniques presented in this thesis were implemented using the Matlab software. If this software is very practical for exploration (due to the intuitive interface and to the fact that Matlab is interpreted rather than compiled), the execution of programs for many computer simulations is usually slower than when using a compiled language, such as the C or C++. Moreover, compiled languages are well adapted for integration into embedded hardware. The resources needed by each technique were studied from a theoretical point of view, although the specific hardware platform would have a significant impact on execution time.

ANNEX 1

Chart parsers

With the exception of IO and VS, all the techniques presented and compared in this thesis makes use of a chart parser during a pre-processing phase, in order to accelerate the re-estimation process of SCFG production rule probabilities. This annex presents the two main existing chart parsers, namely the Earley parser (Earley, 1970) and the Cocke-Younger-Kasami (CYK) parser (Nijholt, 1991; Hopcroft *et al.*, 2001). They also belong to two different classes of parsers: the Earley parser is a top-down parser, that starts from the $Start$ symbol of the grammar to find the rules leading to the sequences, while the CYK parser is bottom-up, and starts from the sequences to get back to the $Start$ symbol of the SCFG. Moreover, the Earley parser does not require the grammar to be in CNF, while the CYK parser does.

1.1 The Earley parser

Given a sequence $w_1, ..., w_n$ as input, an Earley parser will construct a sequence of sets $S_0, ..., S_n$ called Earley sets. Each set S_i is composed of items composed of three parts : a grammar rule $A \rightarrow \lambda\mu$ (in which $\{\lambda, \mu\} \in (V \cup N)^*$), a dot $\bullet$ that indicates the progress in the rule and a pointer j to a previous set. It is usually written as follows, allowing to represent all the information (the number of the set $i = 0, ..., n$, the pointer $j = 0, ..., i-1$, the rule $A \rightarrow \lambda\mu$, and the dot $\bullet$) in one expression:

$$_j^i A \rightarrow \lambda \bullet \mu$$

S_0 is initialized using an additional start symbol S' that allows beginning with $S_0 =_0^0 S' \rightarrow \bullet Start$. Then the three following steps are executed until S_0 does not change :

a. SCANNER: if $_j^i A \rightarrow ... \bullet a...$ is in S_i and $a = w_{i+1}$, add $_j^{i+1} A \rightarrow ...a \bullet ...$ to S_{i+1};

b. PREDICTOR: If $_j^i A \rightarrow ... \bullet B...$ is in S_i, add $_i^i B \rightarrow \bullet\lambda$ to S_i for all rules $B \rightarrow \lambda$;

c. COMPLETER: If ${}^{i}_{j}A \rightarrow ...\bullet$ is in S_i, add ${}^{i}_{k}B \rightarrow ...A \bullet ...$ to S_i for all items ${}^{j}_{k}B \rightarrow ... \bullet A...$ in S_j.

It is important to note here that an item can only appear once in a state. If an item already exists in a set, it should not be added a second time (Alg. 28).

Algorithm 28: `Earley-Parser()`

```
for i=1 to L+1 do
    size1 = |S_i|;
    apply PREDICTOR from the beginning of S_i;
    apply COMPLETER from the beginning of S_i;
    size2 = |S_i|;
    while size2>size1 do
        start = size1+1;
        size1 = size2;
        apply PREDICTOR from start;
        apply COMPLETER from start;
        size2 = |S_i|;
    if i<L+1 then
        apply SCANNER from the beginning of S_i;
```

The main weakness of this algorithm is that it does not support null production. A non-terminal is said to be nullable if it does not emit any combination of terminals and non-terminals[1]. Aycock and Horspool (2002) propose a modification to solve this problem that only consists in modifying the PREDICTOR steps the following way:

> If ${}^{i}_{j}A \rightarrow ... \bullet B...$ is in S_i, add ${}^{i}_{i}B \rightarrow \bullet\lambda$ to S_i for all rules $B \rightarrow \lambda$. **If B is nullable, also add** ${}^{i}_{j}A \rightarrow ...B \bullet ...$ **to** S_i**.**

The next step is to extract from the Earley sets the rules that lead to the parsed sequence. Alg. 29 allows it. In this algorithm, write an item of S_i as $S_i\{k\} = \{[A, B, C], d, j\}$ for a transition rule and $S_i\{k\} = \{[A, a], d, j\}$ for an emission rule. So $S_i\{k, 1\} =$

[1] Note that this situation does not appear in this thesis, but may nevertheless be encountered in a radar ES application.

$[A, B, C]$ *or* $[A, a]$, that refers to the rules $A \rightarrow BC$ and $A \rightarrow a$, $S_i\{k, 2\} = d$, that is the position of the dot and $S_k\{j, 3\} = j$, that is the pointer to a previous set. k is the number of the item of thew current set i.

Algorithm 29: `Earley-Decode()`

```
for i=|S| to 1 do
    while cond2 > cond1 do
        cond1 = cond2;
        for k=1 to |S_i| do
            if ∃ B ∈ ref \ S_i{k,1}(1) = B and "dot at the end of the production rule"
            then
                add S_i{k} to store;
                S_i{k} = ∅;
                cond2 = cond2 + 1;
```

Below is given an example of the Earley parser. The rules of the grammars are $A \rightarrow AaA|b$. The input sequence is $x = bab$. The items of the final derivation are in bold.

$\underline{S_0}$**:**

$S' \rightarrow \bullet A, 0$

Predictor adds: $A \rightarrow \bullet AaA, 0$

Predictor adds: $A \rightarrow \bullet b, 0$

$\underline{S_1}$**:**

Scanner adds: A $\rightarrow$ b$\bullet$, 0

Completer adds: $S' \rightarrow A\bullet, 0$

Completer adds: $A \rightarrow A \bullet aA, 0$

$\underline{S_2}$**:**

Scanner adds: $A \rightarrow Aa \bullet A, 0$

Predictor adds $A \rightarrow \bullet AaA, 2$

Predictor adds: $A \rightarrow \bullet b, 2$

$\underline{S_3}$**:**

Scanner adds: A $\rightarrow$ b$\bullet$, 2

Completer adds: A $\rightarrow$ AaA$\bullet$, 0

Completer adds: $A \rightarrow A \bullet aA, 2$

Completer adds: $\mathbf{S' \rightarrow A\bullet, 0}$

1.2 The CYK parser

There are several ways of presenting the results of the CYK parser. The tabular chart version of the parser will be shown here. Usually, a CYK parser is applied to a grammar in CNF. For a sequence of size L, the results are presented in an upper-triangular $(L+1) \cdot (L+1)$ table Tc, also called chart. A rule in the cell $chart(i, j)$ of the table means that this rule leads to the subsequence $w_i, ..., w_j$, and the first index represent the row. An example of a CYK chart is given in Fig. 10 in Section 3.2.2. The pseudo-code is given in Alg. 30.

Algorithm 30: `CYK-Parser()`

```
%%Initialization: fill the diagonal%%
for i = 0 to L − 1 do
    foreach non-terminal A leading to w_i do
        add A(i, i+1) → w_i(i, i+1) to chart_{i,i+1}
%%Filling%%
for i=L-1 to 0 do
    for j=i+2 to L do
        for k=i+1 to j-1 do
            for m1=1 to |chart_{i,k}| do
                for m2=1 to |chart_{k,j}| do
                    let B be the non-terminal producing chart_{i,k}(m1) and C the
                    non-terminal producing chart_{k,j}(m2);
                    foreach non-terminal A leading to BC do
                        add A(i, j) → B(i, k)C(k, j) to chart_{i,j}
```

ANNEX 2

Non-Implemented Algorithms Based on ML

Following an extensive literature review and preliminary study, only the algorithms presented in Chapters 3 and 4 were retained for further study. However, other algorithms, like k-VS, VS with prior information, Stolcke's method and its extension by Ra and Stockman, may be of interest for the reader. This annex presents a description of these algorithms.

2.1 K-best Derivations (kVS) algorithms

The K-best derivations algorithm (kVS) (Nevado *et al.*, 2000; Sanchez and Benedi, 1999b) is an alternative to the IO and the VS algorithms. While the IO algorithm re-estimates the probabilities by using all the possible parse trees, and the VS algorithm by using only the best parse tree, the k-best derivations algorithm will use the k best parse trees, were k is a a user-defined parameter. $k = 1$ is equivalent to VS.

Since it appeared very soon that classical approaches based on naive implementation, such as IO and VS, cannot be used for radar ES applications, kVS has not been implemented. However, this algorithm can be easily derived from the TS algorithm, and a pseudo-code allowing this is given in Algorithm 31. Adaptation of this algorithm to gEM may be feasible, but is not trivial and was not explored.

2.2 VS algorithm with prior information

Since a MFR system will generate words according to its internal states, a SCFG may be constructed such that these states correspond to the nonterminals that may produced by the start symbol. It is possible for each word of the database to attain the corresponding state. This algorithm no longer selects the most probable tree (as with VS). It instead selects the most probable tree that contains the radar's states corresponding to reality. The reestimation equation will remain the same as Eq. 3.12, but with a different consideration for the path.

Algorithm 31: `Tree-Scanning(kVS)`

for *x=1 to* $|\Omega|$ **do**
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;
  p'_x = rank p_x from the maximum to the minimum;
  d'_x = rank d_x in the same order as p';
while $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  for *x=1 to* $|\Omega|$ **do**
    %%Histograms%%
    initialize $histo\_t_x$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
    initialize $histo\_e_x$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
    for *i=1 to* $|d_x|$ **do**
      for $j = 1$ *to* k **do**
        if $|{d'_x}^i(j)|$*=3* **then**
          $histo\_t_x({d'_x}^i(j)) = histo\_t_x({d'_x}^i(j)) + 1$;
        else
          $histo\_e_x({d'_x}^i(j)) = histo\_e_x({d'_x}^i(j)) + 1$;
    %%Reestimation%%
    $ptotal_x = \sum_{i=1}^{k} p'_i$ $prod\_t_x = \sum_{i=1}^{k} p'_{x,i} \cdot histo\_t_i$;
    $prod\_e_x = \sum_{i=1}^{k} p'_{x,i} \cdot histo\_e_i$;
  $num\_t = \sum_x \frac{prod\_t_x}{ptotal_x}$;
  $num\_e = \sum_x \frac{prod\_e_x}{ptotal_x}$;
  for $i = 1$ *to* M_{nt} **do**
    $denom(i) = \sum_{j=1}^{M_{nt}} \sum_{k=1}^{M_{nt}} num\_t(i,j,k) + \sum_{j=1}^{M_t} num\_e(i,j)$;
  foreach *rule* $A \rightarrow BC$ *or* $A \rightarrow a$ **do**
    $\theta'(A \rightarrow BC) = \frac{num\_t(A,B,C)}{denom(A)}$;
    $\theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}$;
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_i = p_i \cdot \theta(d_x^i(j))$;
  p' = rank p from the maximum to the minimum;
  d' = rank d in the same order as p';
  $cond^{(m)} = \sum_{i=1}^{k} p'_i$;
  $m = m + 1$;

This algorithm is not suitable for very long sequences or very complex grammars, and was only implemented for a phrase-level grammar, that can actually be seen as the grammars presented in Annex 3 applied to phrases only instead of long sequences. This modified

version is given in Algorithm 32. Another version derived from TS and appliable to long sequences is given in Algorithm 33. Adaptation of this algorithm to gEM may be feasible, but is not trivial and was not explored.

Algorithm 32: `VS-prior-information()`

%%Initialization%%
for *i=1 to L* **do**
  for $A \in N$ **do**
    $\alpha'_{max}(i,i|A) = e_A(Wi)$;
%%Iteration%%
for *j=2 to L* **do**
  for *i=j-1 to 1* **do**
    for $A \in N$ **do**
      if *A=Start or* σ_i **then**
        if $j - i <$ *size of a phrase* **then**
          $\alpha'_{max}(j,i|A) = \max_{1\leq B,C\leq M} \max_{i\leq k<j}\{\theta(A \rightarrow state(i)\epsilon)\alpha'_{max}(i,k|state(i))\alpha'_{max}(k+1,j|\epsilon)\}$;
          $\psi'(j,i|A) = \text{argmax}_{B,C,k}\{\theta(A \rightarrow state(i)\epsilon)\alpha'_{max}(i,k|state(i))\alpha'_{max}(k+1,j|1)\}$;
        else
          $\alpha'_{max}(j,i|A) = \max_{1\leq B,C\leq M} \max_{i\leq k<j}\{\theta(A \rightarrow state(i)1)\alpha'_{max}(i,k|state(i))\alpha'_{max}(k+1,j|1)\}$;
          $\psi'(j,i|A) = [B,C,k] = \text{argmax}_{B,C,k}\{t_A(state(i),1)\alpha'_{max}(i,k|state(i))\alpha'_{max}(k+1,j|1)\}$;
      else
        $\alpha'_{max}(j,i|A) = max_{1\leq B,C\leq M} max_{i\leq k<j}\{t_A(B,C)\alpha'_{max}(i,k|B)\alpha'_{max}(k+1,j|C)\}$;
        $\psi'(j,i|A) = [B,C,k] = \text{argmax}_{B,C,k}\{t_A(B,C)\alpha'_{max}(i,k|B)\alpha'_{max}(k+1,j|C)\}$;

2.3 Stolcke's method and Ra's extension

Stolcke (1995) developed an algorithm that was later improved by Ra and Stockman (1999) based on the Earley parser. Stolcke's approach consists in computing inner and outer probabilities by updating them at each step of the Earley algorithm, which requires two passes. Then, the probabilities of the grammar are re-estimated according to Eq. 3.5. An inner probability is of the form $\alpha'({}_i^j S' \rightarrow \lambda \bullet \mu)$, where ${}_i^j A \rightarrow \lambda \bullet \mu$ is an Earley item.

Algorithm 33: `Tree-Scanning(VS-prior-information)`

for *x=1 to* $|\Omega|$ **do**
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;

while $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  %%Histograms%%
  for *x=1 to* $|\Omega|$ **do**
    initialize $histo\_t_x$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
    initialize $histo\_e_x$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
    for *i=1 to* $|d_x|$ **do**
      if d_x^i *contains the radar's states corresponding to reality* **then**
        $dselect_x = d_x^i$;
        $pselect = p_i$;
    for $j = 1$ *to* $|dselect_x|$ **do**
      if $|dselect_x^i(j)|$*=3* **then**
        $histo\_t_x(dselect_x(j)) = histo\_t_x(dselect_x(j)) + 1$;
      else
        $histo\_e_x(dselect_x(j)) = histo\_e_x(dselect_x(j)) + 1$;
  %%Reestimation%%
  $num\_t = \sum_x histo\_t_x$;
  $num\_e = \sum_x histo\_e_x$;
  for $i = 1$ *to* M_{nt} **do**
    $denom(i) = \sum_{j=1}^{M_{nt}} \sum_{k=1}^{M_{nt}} num\_t(i,j,k) + \sum_{j=1}^{M_t} num\_e(i,j)$;
  foreach *rule* $A \rightarrow BC$ *or* $A \rightarrow a$ **do**
    $\theta'(A \rightarrow BC) = \frac{num\_t(A,B,C)}{denom(A)}$;
    $\theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}$;
  $cond^{(m)} = 0$;
  for *x=1 to* $|\Omega|$ **do**
    %%Probabilities computation for each tree%%
    $p_{x,1to|d_x|} = 1$;
    for *i=1 to* $|d_x|$ **do**
      for *j=1 to* $|d_x^i|$ **do**
        $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;
    $cond^{(m)} = cond^{(m)} + \max_i\{p_{x,i}\}$;
  $m = m + 1$;

Inner probabilities give an inside probability according to Eq. 2.1.

$$\alpha(i,j|A) = \sum_{\lambda} \alpha'({}_i^j A \rightarrow \lambda\bullet) \tag{2.1}$$

On the same way, an outer probability is of the form $\beta({}_i^j S' \rightarrow \lambda \bullet \mu)$. Outer probabilities give an outside probability according to Eq. 2.2.

$$\beta(i,j|A) = \sum_{\lambda} \beta'({}_i^j A \rightarrow \lambda\bullet) \tag{2.2}$$

The approach by Ra and Stockman (1999) does not need outside probability but for each item computes the weighted usage of each rule in the same pass that for computing the inside probabilities, and thus re-estimates the probabilities differently. A weighted usage of a rule $B \rightarrow \nu$ is therefore written $\xi({}_i^j A \rightarrow \lambda \bullet \mu, B \rightarrow \nu)$. So for each Earley item, there are as many ξ as there are production rules.

The algorithm can be described according to the following steps:

a. each time the Earley parser completes a step, update the inside probability and the weighted usage of the corresponding items that have been created using the previous items;

b. keep only the total inside probability and the corresponding weighted usage of the rules;

c. re-estimate the probabilities of the rules.

As the Earley parser takes no in account the probabilities values (consider them only as null or non null), the results of the Earley parser would give the same result for a grammar, even if its probabilities are modified. So by storing the operations of the Earley parser, it is possible to apply this algorithm without parsing the sentences at each iteration. In this

case, even if an operation leads to an item that already exists in the same state, it should be added anyway. But if an item appears more than once, only the first should be expanded, as in the normal Earley parsing. So during an Earley parsing, two sequences of sets $S_0, ..., S_n$ (as normally) and $S'_0, ..., S'_n$ should be constructed. The parsing should be constructed on the first sequence of sets in which an item can only appear once in a set, and the second sequence stores all the items even if they appear more than once in a set.

The algorithm can therefore be described according to the following steps:

a. apply the Earley parser on the sentences that create $S_0, ..., S_n$ and $S'_0, ..., S'_n$;

b. for every Earley item of every Earley set $S_0, ..., S_n$, update the corresponding inside probability and weighted usage;

c. keep only the total inside probability and the corresponding weighted usage of the rules;

d. re-estimate the probabilities of the rules.

Algorithm 34: `RA ( )`

for $x \in \Omega|$ **do**
  $(\alpha'({}_0^n S' \rightarrow Start\bullet), \xi({}_0^n S' \rightarrow Start\bullet, r)) = Compute - Elements$;
  for *each rule* r **do**
    $\phi^x(r) = \frac{\xi({}_0^n S' \rightarrow Start\bullet, r)}{\alpha'({}_0^n S' \rightarrow Start\bullet)}$;
for A=*1 to* M_t **do**
  for B=*1 to* M_t **do**
    for C=*1 to* M_t **do**
      $t_A(B,C) = \frac{\sum_{x \in \Omega} \phi^x(A \rightarrow BC)}{\sum_{x \in \Omega} \sum_B \sum_C \phi^x(A \rightarrow BC)}$;
  for a=*1 to* N **do**
    $e_A(a) = \frac{\sum_{x \in \Omega} \phi^x(A \rightarrow a)}{\sum_{x \in \Omega} \sum_a \phi^x(A \rightarrow a)}$;

Algorithm 35: `Compute-Elements()`

for *i=1 to n* **do**
  for *j=1 to* $|S_i|$ **do**
    if *the item* u *comes from a PREDICTOR operation* **then**
      if u *is of the form* ${}^{s}_{s}A \rightarrow \bullet\lambda$ **then**
        $\alpha'(u) = P(A \rightarrow \lambda)$;
        for *each rule* r **do**
          $\xi(u, r) = 0$;
    if *the item* u_2 *comes from a SCANNER operation* **then**
      if u_1 *leads to* u_2 **then**
        $\alpha'(u_2) = \alpha'(u_1)$;
        for *each rule* r **do**
          $\xi(u2, r) = \xi(u1, r)$;
    if *the item* u_3 *comes from a COMPLETER operation* **then**
      if $u_3 =^{t}_{s} A \rightarrow \lambda B \bullet \mu$ *is produced by* $u_1 =^{e}_{s} A \rightarrow \lambda \bullet B\mu$ *and* $u_2 =^{t}_{e} B \rightarrow \sigma\bullet$ **then**
        if u_3 *does not already exists* **then**
          first set $\alpha'(u_3) = 0$;
          for *each rule* r **do**
            first set $\xi(u_3, r) = 0$;
        $\alpha'(u_3) = \alpha'(u_3) + \alpha'(u_1) \cdot \alpha'(u_2)$;
        for *each rule* r **do**
          if $r \neq B \rightarrow \sigma$ **then**
            $\xi(u_3, r) = \xi(u_3, r) + \xi(u_1, r) \cdot \alpha'(u_2, r) + \xi(u_2, r) \cdot \alpha'(u_1, r)$;
          else
            $\xi(u_3, r) =$
            $\xi(u_3, r) + \xi(u_1, r) \cdot \alpha'(u_2, r) + \xi(u_2, r) \cdot \alpha'(u_1, r) + \alpha'(u_1, r) \cdot \alpha'(u_2, r)$;

In the following u_i or u will represent an Earley item ${}^{t}_{s}A \rightarrow \lambda \bullet \mu$, where t is the current number of the Earley state and s is the pointer to a previous set, and $r = A \rightarrow \lambda$ a rule of the grammar. Supposing $S_0, ..., S_n$ have been created, the approach is described in algorithms 35 and 34. The relation between the elements of the algorithm and those of Eq. 3.2 can be expressed as follows. For each sequence x of the database, set $\phi^x(A \rightarrow \lambda)$ the balanced frequency of the rule $A \rightarrow \lambda$:

$$\phi^x(A \rightarrow \lambda) = \frac{1}{P(x, \Delta_x|G)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x|G) \tag{2.3}$$

Then according to Eq. 3.2 the reestimation of the probabilities can be written as follows:

$$\theta(A \rightarrow \lambda) = \frac{\sum_{x \in \Omega} \phi^x(A \rightarrow \lambda)}{\sum_{x \in \Omega} \sum_{\lambda} \phi^x(A \rightarrow \lambda)} \tag{2.4}$$

Ra and Stockman (1999) have shown that $\phi^x(A \rightarrow \lambda)$ can be obtained using:

$$\phi^x(A \rightarrow \lambda) = \frac{\xi({}^n_0 S' \rightarrow Start\bullet, A \rightarrow \lambda)}{\alpha'({}^n_0 S' \rightarrow Start\bullet)} \tag{2.5}$$

This algorithm gives the same result as the IO algorithm. An Earley item is obtained using the combination of others items, and the quantities α' and ξ also result from this combination. The values of $\xi({}^n_0 S' \rightarrow Start\bullet, A \rightarrow \lambda)$ and $\alpha'({}^n_0 S' \rightarrow Start\bullet)$ (in Eq. 2.5) are obtained using a path of combinations of Earley items which is usually less complex than using all the combinations of nonterminals as in the classical IO algorithm. In the average cases, their time complexity per iteration is lower than that of IO.

Since the MFR grammars are too large to allow demonstrating this algorithm, its operation is only illustrated using an example given by Aycock and Horspool (2002). The rules of the grammar in this example are $A \rightarrow AaA|b$ with probabilities of 0.75 and 0.25. The input sequence is $x = bab$. In this case the Earley parser contains redundant items, as they have to be considered in the algorithm, and the values of α and ξ are given by (the bold step indicates the elements to use for re-estimation):

$\underline{S'_0}$:

$S' \rightarrow \bullet A, 0$

$\alpha'({}^0_0 S' \rightarrow \bullet A) = 1$

$\xi({}_0^0 S' \rightarrow \bullet A, A \rightarrow AaA) = 0$ and $\xi({}_0^0 S' \rightarrow \bullet A, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet AaA, 0$

$\alpha'({}_0^0 A \rightarrow \bullet AaA) = 0.75$

$\xi({}_0^0 A \rightarrow \bullet AaA, A \rightarrow AaA) = 0$ and $\xi({}_0^0 A \rightarrow \bullet AaA, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet b, 0$

$\alpha'({}_0^0 A \rightarrow \bullet b) = 0.25$

$\xi({}_0^0 A \rightarrow \bullet b, A \rightarrow AaA) = 0$ and $\xi({}_0^0 A \rightarrow \bullet b, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet AaA, 0$

$\alpha'({}_0^0 A \rightarrow \bullet AaA) = 0.75$

$\xi({}_0^0 A \rightarrow \bullet AaA, A \rightarrow AaA) = 0$ and $\xi({}_0^0 A \rightarrow \bullet AaA, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet b, 0$

$\alpha'({}_0^0 A \rightarrow \bullet b) = 0.75$

$\xi({}_0^0 A \rightarrow \bullet b, A \rightarrow AaA) = 0$ and $\xi({}_0^0 A \rightarrow \bullet b, A \rightarrow b) = 0$

$\underline{S_1'}$:

Scanner adds: $A \rightarrow b\bullet, 0$

$\alpha'({}_0^1 A \rightarrow b\bullet) = 0.25$

$\xi({}_0^1 A \rightarrow b\bullet, A \rightarrow AaA) = 0$ and $\xi({}_0^1 A \rightarrow b\bullet, A \rightarrow b) = 0$

Completer adds: $S' \rightarrow A\bullet, 0$

$\alpha'({}_0^1 S' \rightarrow A\bullet) = 0.25$

$\xi({}_0^1 S' \rightarrow A\bullet, A \rightarrow AaA) = 0$ and $\xi({}_0^1 S' \rightarrow A\bullet, A \rightarrow b) = 0$

Completer adds: $A \rightarrow A \bullet aA, 0$

$\alpha'({}_0^1 A \rightarrow A \bullet aA) = 0.1875$

$\xi({}_0^1 A \rightarrow A\bullet aA, A \rightarrow AaA) = 0$ and $\xi({}_0^1 A \rightarrow A\bullet aA, A \rightarrow b) = 0.1875$

$\underline{S_2'}$:

Scanner adds: $A \rightarrow Aa \bullet A, 0$

$\alpha'({}_0^2 A \rightarrow AaA) = 0.1875$

$\xi({}_0^2 A \rightarrow Aa\bullet A, A \rightarrow AaA) = 0$ and $\xi({}_0^2 A \rightarrow Aa\bullet A, A \rightarrow b) = 0.1875$

Predictor adds $A \rightarrow \bullet AaA, 2$

$\alpha'({}_2^2 A \rightarrow AaA) = 0.75$

$\xi({}_2^2 A \rightarrow \bullet AaA, A \rightarrow AaA) = 0$ and $\xi({}_2^2 A \rightarrow \bullet AaA, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet b, 2$

$\alpha'({}_2^2 A \rightarrow b) = 0.25$

$\xi({}_2^2A \rightarrow \bullet b, A \rightarrow AaA) = 0$ and $\xi({}_2^2A \rightarrow \bullet b, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet AaA, 2$

$\alpha'({}_2^2A \rightarrow \bullet AaA) = 0.75$

$\xi({}_2^2A \rightarrow \bullet AaA, A \rightarrow AaA) = 0$ and $\xi({}_2^2A \rightarrow \bullet AaA, A \rightarrow b) = 0$

Predictor adds: $A \rightarrow \bullet b, 2$

$\alpha'({}_2^2A \rightarrow \bullet b) = 0.25$

$\xi({}_2^2A \rightarrow \bullet b, A \rightarrow AaA) = 0$ and $\xi({}_2^2A \rightarrow \bullet b, A \rightarrow b) = 0$

$\underline{S'_3}$:

Scanner adds: $A \rightarrow b\bullet, 2$

$\alpha'({}_2^3A \rightarrow b\bullet) = 0.25$

$\xi({}_2^3A \rightarrow b\bullet, A \rightarrow AaA) = 0$ and $\xi({}_2^3A \rightarrow b\bullet, A \rightarrow b) = 0$

Completer adds: $A \rightarrow AaA\bullet, 0$

$\alpha'({}_0^3A \rightarrow AaA\bullet) = 0.046875$

$\xi({}_0^3A \rightarrow AaA\bullet, A \rightarrow AaA) = 0$ and $\xi({}_0^3A \rightarrow AaA\bullet, A \rightarrow b) = 0.09375$

Completer adds: $A \rightarrow A \bullet aA, 2$

$\alpha'({}_2^3A \rightarrow A \bullet aA) = 0.18753$

$\xi({}_2^3A \rightarrow A\bullet aA, A \rightarrow AaA) = 0$ and $\xi({}_2^3A \rightarrow A\bullet aA, A \rightarrow b) = 0.1875$

Completer adds: $\mathbf{S' \rightarrow A\bullet, 0}$

$\alpha'({}_0^3\mathbf{S' \rightarrow A\bullet}) = \mathbf{0.046875}$

$\xi({}_0^3\mathbf{S' \rightarrow A\bullet, A \rightarrow AaA}) = \mathbf{0.046875}$

and $\xi({}_0^3\mathbf{S' \rightarrow A\bullet, A \rightarrow b}) = \mathbf{0.09375}$

Completer adds: $A \rightarrow A \bullet aA, 0$

$\alpha'({}_0^3A \rightarrow A \bullet aA) = 0.0351563$

$\xi({}_0^3A \rightarrow A \bullet aA, A \rightarrow AaA) = 0.0351563$ and $\xi({}_0^3A \rightarrow A \bullet aA, A \rightarrow b) = 0$

This result allows to compute the balanced frequencies ϕ^x:

$$\phi^x(A \rightarrow AaA) = \frac{\xi({}_0^3S' \rightarrow A\bullet, A \rightarrow AaA)}{\alpha'({}_0^3S' \rightarrow A\bullet)} = \frac{0.046875}{0.046875} = 1$$

$$\phi^x(A \rightarrow b) = \frac{\xi({}_0^3S' \rightarrow A\bullet, A \rightarrow b)}{\alpha'({}_0^3S' \rightarrow A\bullet)} = \frac{0.09375}{0.046875} = 2$$

Here the values are exactly 1 and 2 because there is only one sequence as an input. The probabilities are re-estimated as follows:

$$\theta(A \rightarrow AaA) = \frac{\phi^x(A \rightarrow AaA)}{\phi^x(A \rightarrow AaA) + \phi^x(A \rightarrow b)} = \frac{1}{3}$$

$$\theta(A \rightarrow b) = \frac{\phi^x(A \rightarrow b)}{\phi^x(A \rightarrow AaA) + \phi^x(A \rightarrow b)} = \frac{2}{3}$$

Which corresponds to the derivation tree of Fig. 28.

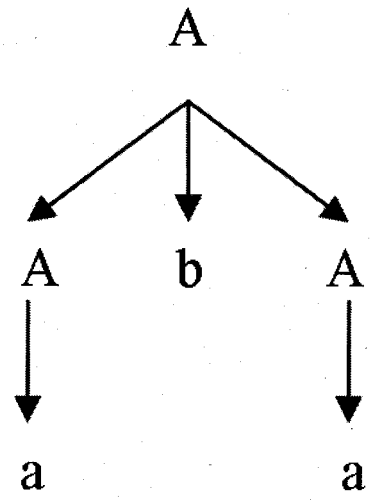


Figure 28 Derivation tree of the sequence AaA

ANNEX 3

Languages and Gramars of MFRs

This annex presents the grammars, languages, and parameters of state duration associated with the Mercury and Pluto MFRs. It also provides the languages associated with the Venus MFRs named VenusA and VenusB.

3.1 Mercury

3.1.1 Mercury grammar

$Start \rightarrow Search\ \sigma 1 \mid Acq\ \sigma 2 \mid Na\ \sigma 3 \mid Rr\ \sigma 4 \mid Tm\ \sigma 5 \mid$

$Search\ \epsilon \mid Acq\ \epsilon \mid Na\ \epsilon \mid Rr\ \epsilon \mid Tm\ \epsilon$

$\sigma 1 \rightarrow Search\ \sigma 1 \mid Acq\ \sigma 2 \mid Search\ \epsilon \mid Acq\ \epsilon$

$\sigma 2 \rightarrow Acq\ \sigma 2 \mid Na\ \sigma 3 \mid Acq\ \epsilon \mid Na\ \epsilon$

$\sigma 3 \rightarrow Na\ \sigma 3 \mid Rr\ \sigma 4 \mid Na\ \epsilon \mid Rr\ \epsilon$

$\sigma 4 \rightarrow Rr\ \sigma 4 \mid Tm\ \sigma 5 \mid Rr\ \epsilon \mid Tm\ \epsilon$

$\sigma 5 \rightarrow Rr\ \sigma 4 \mid Tm\ \sigma 5 \mid Search\ \sigma 1 \mid Rr\ \epsilon \mid Tm\ \epsilon \mid Search\ \epsilon$

$Search \rightarrow W12\ W45 \mid W24\ W51 \mid W45\ W12 \mid W51\ W24 \mid$

$W13\ W51 \mid W35\ W13 \mid W51\ W35$

$Acq \rightarrow Q1\ Q1 \mid Q2\ Q2 \mid Q3\ Q3 \mid Q4\ Q4 \mid Q5\ Q5 \mid Q6\ Q6$

$Na \rightarrow S1\ T6 \mid Q6\ Q6$

$Rr \rightarrow W7\ T6 \mid W8\ T6 \mid W9\ T6$

$Tm \rightarrow Q6\ Q6 \mid Q7\ Q7 \mid Q8\ Q8 \mid Q9\ Q9 \mid S1\ T6 \mid S2\ T7 \mid S2\ T8 \mid S2\ T9$

$W12 \rightarrow W1\ W2$

$W45 \rightarrow W4\,W5$

$W24 \rightarrow W2\,W4$

$W51 \rightarrow W5\,W1$

$W13 \rightarrow W1\,W3$

$W35 \rightarrow W3\,W5$

$S2 \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6$

$S1 \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5$

$Q1 \rightarrow W1\,W1$

$Q2 \rightarrow W2\,W2$

$Q3 \rightarrow W3\,W3$

$Q4 \rightarrow W4\,W4$

$Q5 \rightarrow W5\,W5$

$Q6 \rightarrow W6\,W6$

$Q7 \rightarrow W7\,W7$

$Q8 \rightarrow W8\,W8$

$Q9 \rightarrow W9\,W9$

$T6 \rightarrow Q6\,W6$

$T7 \rightarrow Q7\,W7$

$T8 \rightarrow Q8\,W8$

$T9 \rightarrow Q9\,W9$

$W1 \rightarrow 1$

$W2 \rightarrow 2$

$W3 \rightarrow 3$

$W4 \rightarrow 4$

$W5 \rightarrow 5$

$W6 \rightarrow 6$

$W7 \rightarrow 7$

$W8 \rightarrow 8 \qquad W9 \rightarrow 9$

Where V={1,2,3,4,5,6,7,8,9}, and ϵ is the empty string.

3.1.2 Mercury language

Table XVIII

Mercury language.

State	Phrases	State	Phrases	State	Phrases
S	1245	Acq	1111	Na	6666
	2451		2222	Rr	7666
	4512		3333		8666
	5124		4444		9666
	1351		5555	Tm	6666
	3513		6666		7777
	5135				8888
					9999

3.1.3 Parameters of the states duration for Mercury

Table XIX

Parameters of the states duration for Mercury, in seconds.

	States				
	S	Acq	Na	Rr	Tm
Mean	4	3	1	3	10
Variance	1	1	0.5	0.6	5
Minimum duration	0.5	0.5	0.25	0.6	2
Maximum duration	8	10	3	9	25

3.1.4 The Pluto grammar

$Start \rightarrow Search\,\sigma 1 \mid Na\,\sigma 2 \mid Rr\,\sigma 3 \mid Tm\,\sigma 4$

$\sigma 1 \rightarrow Search\,\sigma 1 \mid Na\,\sigma 2 \mid 7$

$\sigma 2 \rightarrow Na\,\sigma 2 \mid Rr\,\sigma 3 \mid 7$

$\sigma 3 \rightarrow Rr\,\sigma 3 \mid Tm\,\sigma 4 \mid 7$

$\sigma 4 \rightarrow Tm\,\sigma 4 \mid Search\,\sigma 1 \mid 7$

$Search \rightarrow Q2\,W5$

$Na \rightarrow Q2\,W5$

$Rr \rightarrow W0t\,W5$

$Tm \rightarrow W2t\,W5$

$Q2 \rightarrow W2\,W2$

$W2t \rightarrow W2\,W1 \mid W2\,W2 \mid W2\,W3 \mid W2\,W4$

$W0t \rightarrow W0\,W2 \mid W0\,W2 \mid W0\,W3 \mid W0\,W4$

$W0 \rightarrow 0$

$W1 \rightarrow 1$

$W2 \rightarrow 2$

$W3 \rightarrow 3$

$W4 \rightarrow 4$

$W5 \rightarrow 5$

Where V={0,1,2,3,4,5}, and 6 is the empty string.

3.1.5 Pluto language

Table XX

Pluto language.

State	Phrases	State	Phrases
S	2 2 5	Tm	2 1 5
Na	2 2 5		2 2 5
Rr	0 1 5		2 3 5
	0 2 5		2 4 5
	0 3 5		
	0 4 5		

3.1.6 Parameters of the states duration for Pluto

Table XXI

Parameters of the states duration for Pluto, in seconds.

	States			
	S	Na	Rr	Tm
Mean	4	1	1	10
Variance	1	0.5	0.2	5
Minimum duration	0.5	0.25	0.2	2
Maximum duration	8	3	3	25

3.2 VenusA

Table XXII

VenusA language.

State	Phrases	State	Phrases
GS	1 4 3 2 3 4	Tm	phrases of undefined duration consisting in the repetition of the same word (1 to 14)
	4 3 2 3 4 1		
	3 2 3 4 1 4		
	2 3 4 1 4 3		
	3 4 1 4 3 2		
	4 1 4 3 2 3		

3.3 VenusB

Table XXIII

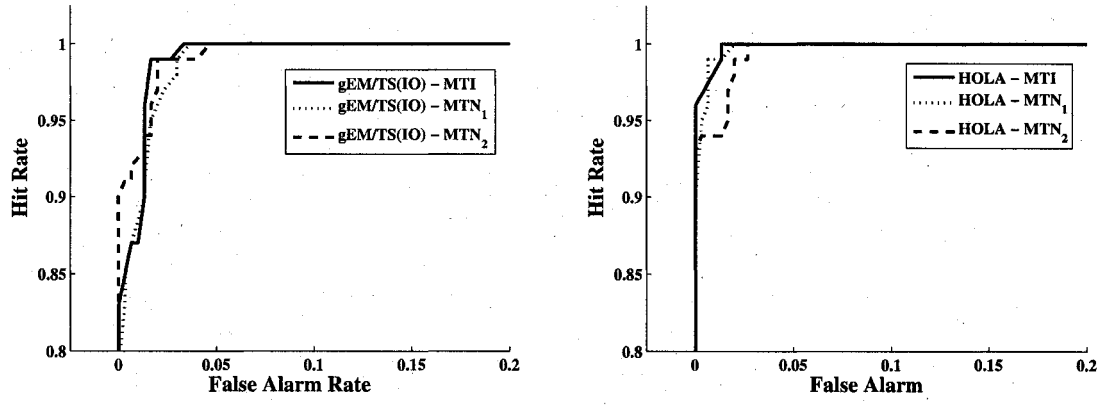
VenusB language.

State	Phrases	State	Phrases
GS	1 2 3	Acq_2	9 7 x1 14 8 x2...
	2 3 1		...11 6 x3 13 9 x4...
	3 1 2		...12 10 x5 7 14 x6...
DS	1 3 4 2		...8 11 x7 6 13 x8...
	3 4 2 1		...9 12 x9
	4 2 1 3		xi = 5...14
	2 1 3 4	Tm	phrases of undefined duration consisting in the repetition of the same word (1 to 14)
Acq_1	one sequence of undefined duration consisting in the repetition of the same word (5 to 14)		

ANNEX 4

Additional results on MFR data

Fig. 29 shows the ROC curves for the Mercury grammar, considering only gEM/TS(IO) and HOLA, using slightly different databases than $\texttt{MROCTN}_1$ and $\texttt{MROCTN}_2$. In the previous case, $\texttt{MROCTN}_1$, $\texttt{MROCTN}_2$ contains noisy sequences emitted by Mercury ($\texttt{MTN}_1$ et $\texttt{MTN}_2$), while the sequences emitted by Pluto, VenusA and VenusB remains unmodified (VATI and VBTI). To compute the new ROC curves, in order to model a situation in which all emitters are noisy, noise was also added to the sequences emitted by VenusA and VenusB the same way as for $\texttt{MTN}_1$ and $\texttt{MTN}_2$. This results in four databases named $\texttt{VATN}_1$, $\texttt{VATN}_2$, $\texttt{VBTN}_1$ and $\texttt{VBTN}_2$. Thus, two new databases were created, named $\texttt{MROCTN}_1^*$ – compounded by $\texttt{MTN}_1$, $\texttt{PTN}_1$, $\texttt{VATN}_1$ and $\texttt{VBTN}_1$ – and $\texttt{MROCTN}_2^*$ – compounded by $\texttt{MTN}_2$, $\texttt{PTN}_2$, $\texttt{VATN}_2$ and $\texttt{VBTN}_2$.



(a) Mercury - gEM/TS(IO): AUC = 0.9980 / 0.9974 / 0.9983

(b) Mercury - HOLA: 0.9997 / 0.9995 / 0.9989

Figure 29 ROC curves obtained for SCFGs trained using (a) gEM/TS(IO); (b)HOLA on MROCTI, $\texttt{MROCTN}_1^*$, and $\texttt{MROCTN}_2^*$.

It can be seen that the level of noise does not have a significant impact on the results. Indeed, since the errors come from the sequences emitted by VenusA and VenusB, if adding noise on sequences from Mercury make these sequecne diverge from the Mercury language, it does the same for the sequences from the two other emitters, and so does not affect the detection. It can be seen that although learning with HOLA leads to a lower

perplexity, it appears to provide a more accurate recognition of emitters. but with an order of magnitude that may not be significant.

ANNEX 5

Experiments on DNA data

In order to confirm the performance of gEM(IO) and gEM(VS), experiments have been led on real world data, composed of DNA sequences from the European Molecular Biology Laboratory (EMBL). Experiments led on MFR data did not show significant difference between these two algorithms. Indeed, even if the Pluto grammar is more ambiguous than Mercury grammar, they both have a limited ambiguity. Since gEM(VS) approximates the likelihood of the best derivation trees of the sequences of the training database, while gEM(IO) approximates the likelihood of all the derivations trees, differences should appear more clearly if using more ambiguous grammars, which is the case for the experiments presented in this annex. Indeed, DNA sequences are very more complex than MFR sequences, and therefore require more ambiguous grammars to be modeled. In a first time, this section details the experimental methodology: it describes the data, the experimental protocol and gives the grammars used in these experiments. In a second part, the results are exposed and discussed. This work also has been presented in (Latombe *et al.*, 2006e).

5.1 Experimental methodology

5.1.1 The EMBL Data

Real-world data consisting of human DNA sequences from the European Molecular Biology Laboratory (EMBL) [1] has been selected. The original data set consists of numerous sequences of human DNA of variable sizes, from approximately 1000 to 12000 symbols. Only the sequences corresponding to "Homo sapiens mRNA, complete cds" were employed in this study. Each sequence is a string of the nucleotides A, C, G and T.

In order to observe the influence of the length of the sequences on the performance, the sequences are divided differently to produce two databases with the same information. Each sequence is divided in sub-sequences of 10 nucleotides for the first database and of

[1] ftp://ftp.ebi.ac.uk/pub/databases/embl/release/hum01.dat.gz

20 for the second one. The nucleotides at the end of a sequence that do not form a sub sequence of 10 or 20 elements are discarded. This gives the two following bases:

- DATA 10: 10 words per sequence gives a training set of 1920 sequences, a validation set of 1000 sequences, and a test set of 1000 sequences.
- DATA 20: 20 words per sequence gives a training set of 960 sequences, a validation set of 500 sequences, and a test set of 500 sequences.

From each training set 3 other sets are created by selecting the first 480, 960, 1440 or 1920 sequences for the sequences of 10 nucleotides, and 240, 480, 720 or 960 sequences for the sequences of 20 nucleotides. The sizes of the different database was chosen so that it would contain the same proportion of information for sequences of 10 and 20 nucleotides.

As only one language is used in this study, to observe the influence of the ambiguity on the algorithms, it has been decided to use two grammars that generate the EMBL data with different levels of ambiguity. They are inspired by the work of Dowell and Eddy (2004) and Sakakibara *et al.* (1993). Their purpose is to represent the secondary structure of a RNA sequence, which is directly derived from a DNA sequence. Their grammar was the following:

$$S \rightarrow a\, S\, \hat{a} \mid a\, S \mid S\, a \mid a$$

where a and $\hat{a}$ are nucleotides and S is a non-terminal. It should be noted that this grammar does not allow taking into account the order of the nucleotides. It was therefore modified by considering several nonterminal (as shown in Appendix 5.2). This leads to the first grammar G_H, that has an ambiguity per word of 4, while the second grammar G_L does not consider pairs of nucleotides, and has an ambiguity per word of 2. The ambiguity per word is the number of derivation trees of a sequence divided by the number of words of the sequence.

5.1.2 Experimental protocol

As only the batch versions of gEM are tested on DNA data in this study, only one experimental protocol is needed. During each trial, training was performed using hold out validation until the difference between the negative log likelihoods (for the IO version) or the approximate negative log likelihoods (for the VS version) of the sequences on the validation subset was lower than 0.01 for two consecutive iterations, or until a maximum of 20 iterations is reached. At the end of each trial, a set of SCFG probabilities were kept, corresponding to the minima of the negative log likelihoods or of the approximate negative log likelihoods of the sequences of the validation base. Finally, the performance was assessed using the test subset.

Each simulation trial was repeated for the 10 different initializations of the probabilities. G_H is initialized in 10 different ways: 1 in a uniform way, 5 in a random way, 1 in a uniform way and then weighting the Watson-Crick pairs twice more, 1 in a random way and then weighting the Watson-Crick pairs twice more, 1 in a uniform by part way and 1 in a uniform by part way and then weighting the Watson-Crick pairs twice more. To initialize the probabilities in a uniform by part way, the rules are separated to differentiate $S \rightarrow a\, S\, \hat{a}$, $S \rightarrow a\, S \mid S\, a$ and $S \rightarrow a$. G_L is also initialized in 10 different ways: 1 in a uniform way and 9 in a random way. Average results, with corresponding standard error, are always obtained as a result of the 10 independent simulation trials.

In order to study the capacity of the grammar to differentiate DNA sequences of different species, ROC curves are computed. To create these curves human DNA sequences, invertebrate DNA sequences, and plant DNA sequences were used. They were divided as explained before into subsequences of 10 and 20 nucleotides in order to get to test data sets called `ROCTest10` and `ROCTest20`. `ROCTest10` is composed of 1000 subsequences of each species while `ROCTest20` is composed of 500 subsequences of each species.

This experimental protocol is summarized in Fig. 30.

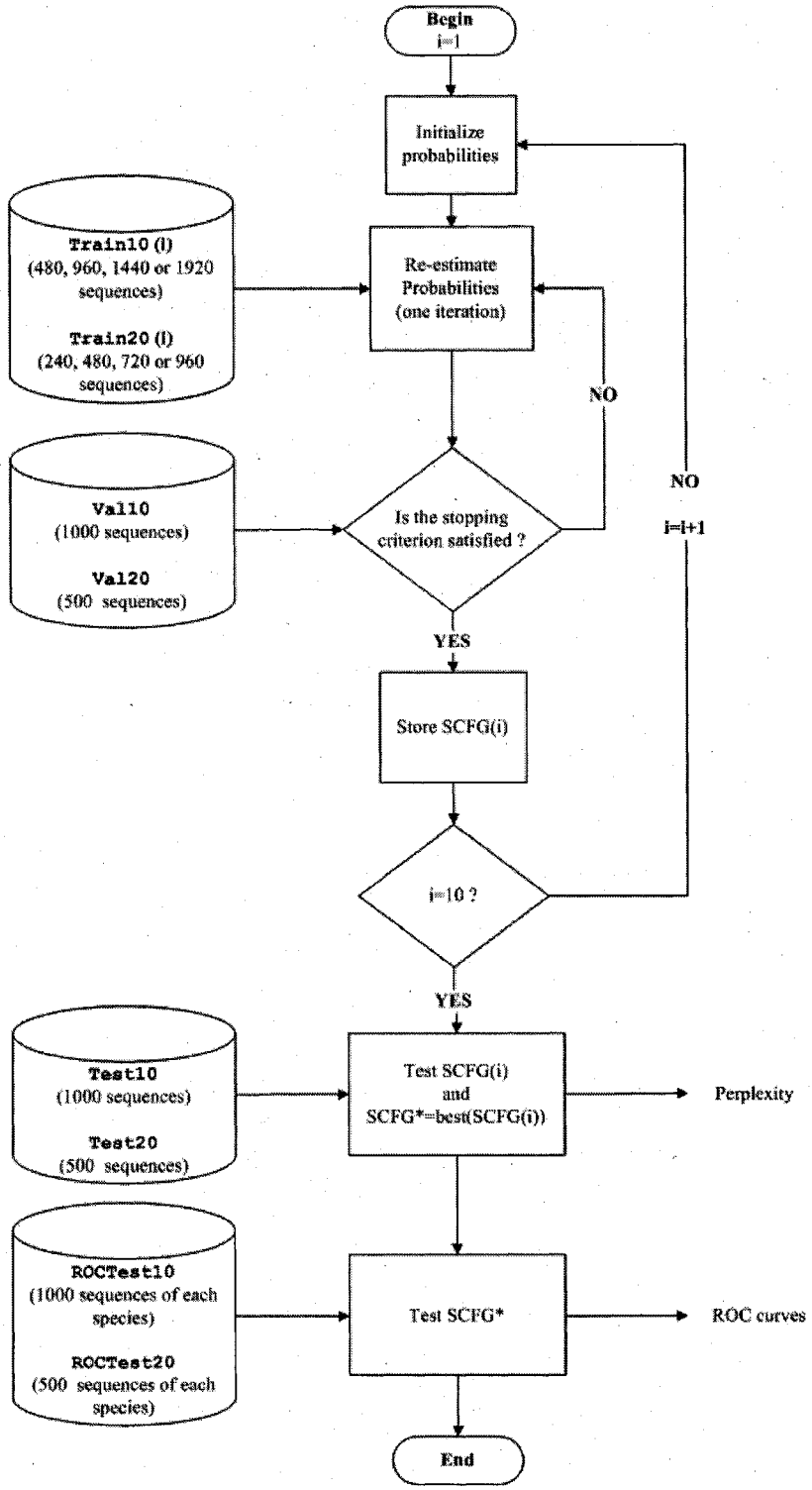


Figure 30 Overall view of experimental protocol for the DNA experiments.

5.2 DNA Grammars

5.2.1 High-Ambiguity Grammar

$Start \mid S1 \mid S2 \mid S3 \mid S4 \mid S5$

$\rightarrow xS1\ A \mid xS2\ G \mid xS3\ C \mid xS2\ T \mid A\ S5 \mid G\ S5 \mid C\ S5$

$\mid T\ S5 \mid S5\ A \mid S5\ G \mid S5\ C \mid S5\ T \mid S6\ S6 \mid a \mid g \mid c \mid t$

$S6 \rightarrow xS1\ A \mid xS2\ G \mid xS3\ C \mid xS2\ T \mid A\ S5 \mid G\ S5 \mid C\ S5$

$\mid T\ S5 \mid S5\ A \mid S5\ G \mid S5\ C \mid S5\ T$

$xS1 \rightarrow A\ S1 \mid G\ S1 \mid C\ S1 \mid T\ S1$

$xS2 \rightarrow A\ S2 \mid G\ S2 \mid C\ S2 \mid T\ S2$

$xS3 \rightarrow A\ S3 \mid G\ S3 \mid C\ S3 \mid T\ S3$

$xS4 \rightarrow A\ S4 \mid G\ S4 \mid C\ S4 \mid T\ S4$

$A \rightarrow a$

$G \rightarrow g$

$C \rightarrow c$

$T \rightarrow t$

5.2.2 Low-Ambiguity Grammar

$Start \mid S1 \mid S2 \mid S3 \mid S4 \mid S5$

$\rightarrow \mid A\ S1 \mid G\ S2 \mid C\ S3 \mid T\ S4 \mid S1\ A \mid S2\ G \mid S3\ C$

$$| S4\,T \mid a \mid g \mid c \mid t$$

$A \rightarrow a$

$G \rightarrow g$

$C \rightarrow c$

$T \rightarrow t$

5.2.3 Results

DNA data was used to compare the performances of gEM(IO) and gEM(VS). That is why only results concerning these two algorithms are given in this section.

Fig. 31 gives the perplexity and approximate perplexity on test data for a SCFG trained using gEM(IO) and gEM(VS) according to the training data set size. This figure indicates that gEM(IO) gives better results in term of perplexity than gEM(VS). However, this is no longer the case when considering the approximate perplexity. This is logical, as gEM(IO) optimizes the perplexity while gEM(VS) optimizes the approximate likelihood.

The size of the training set has little impact on the performances. The results on the approximate perplexity appear to increase with it when gEM(IO) is used for training. This seems strange, cannot be controled, as gEM(IO) optimizes the likelihood over all the parse trees.

Increasing the size of the training sequences makes the perplexity decrease or stay equal, while it makes the approximate perplexity increase when gEM(IO) is used. This happens because more parse trees can be derived from a longer sequence. It can be seen gEM(VS) is not sensitive to this problem. Increasing the size of the training sequences also makes the difference of results between gEM(IO) and gEM(VS) decrease.

Both the perplexity and the approximate perplexity decrease when the level of ambiguity of the grammar increases. Actually, this is due to the fact that G_H is more complex and so can model the data in a better way than G_L. On the other hand it makes the difference of results between gEM(IO) and gEM(VS) increase.

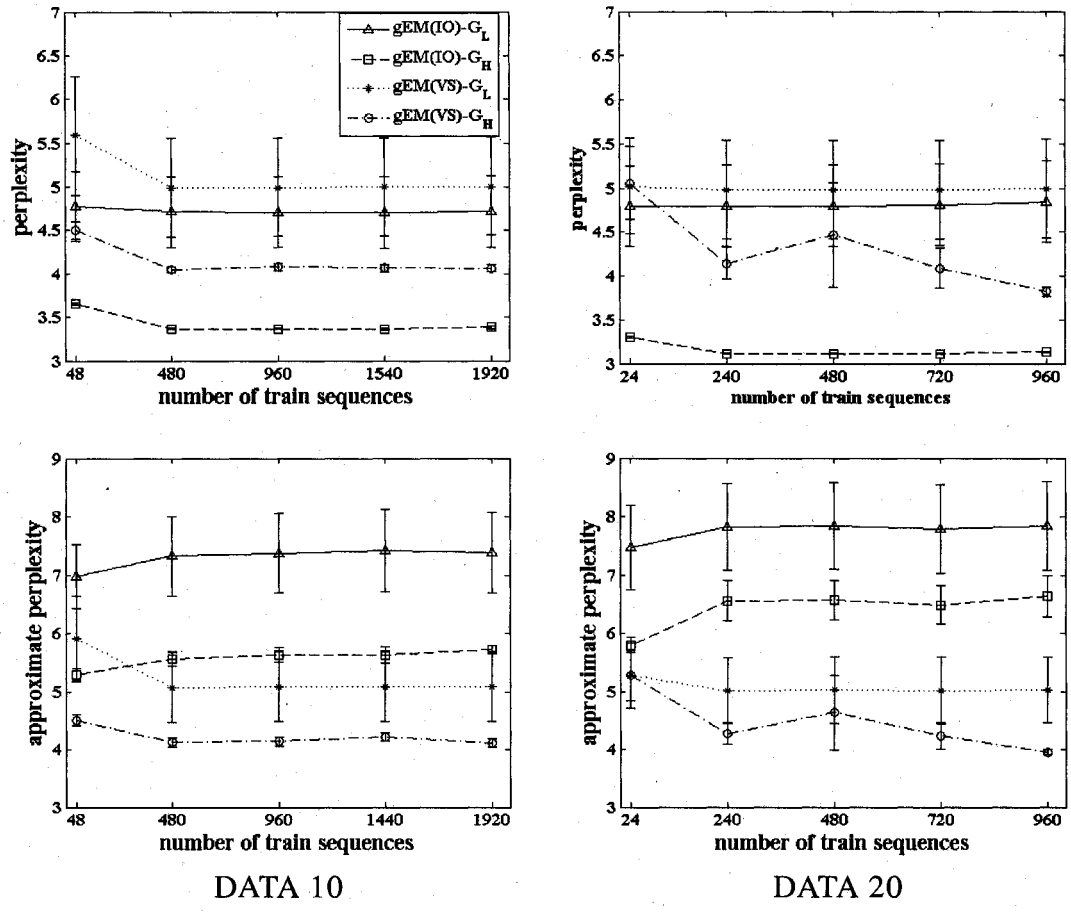


Figure 31 Average perplexity SCFGs of gEM(IO) and gEM(VS) versus size of training set, for DATA 10 and DATA 20. (Error bars are standard error of the data set mean.)

Fig. 32 gives the convergence time needed by the algorithms for DATA 10 and DATA 20 versus the size of the training set.

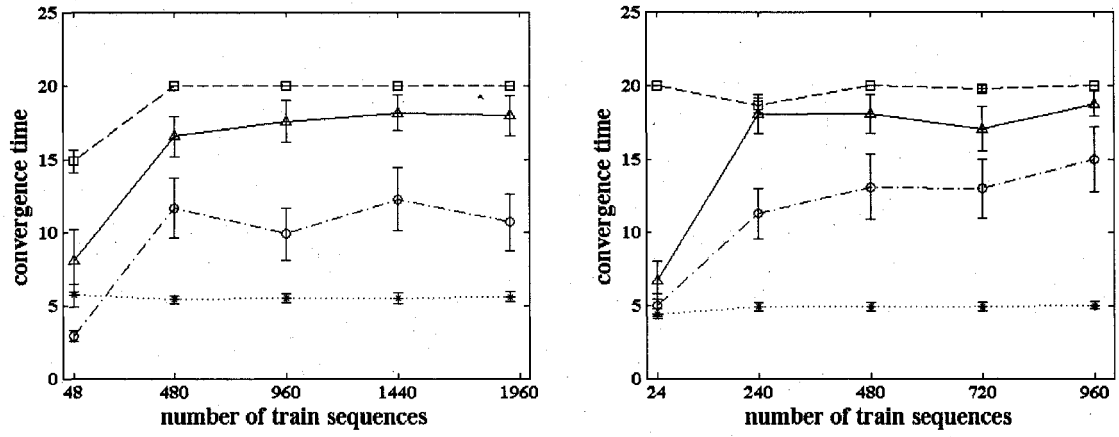


Figure 32 Convergence time for gEM(IO) and gEM(VS) for DATA 10 and DATA 20

The algorithms tend to converge faster for a smaller size of the training set. For both the size of the sequences has little impact on the convergence time while it tends to increase with the degree of ambiguity of the grammar. It can also be seen that gEM(VS) converges faster than gEM(IO), with an approximate ratio of 3 for the low ambiguity grammar and 1.8 for the high ambiguity grammar. However, as a maximum of 20 iterations was allowed during the iterative process, it appears as the maximum number of iterations for gEM(IO) and the high ambiguity grammar. The previous ratio of 1.8 is expected to increase without this limit.

The experimental results on time complexity gave a ratio $T_{gEM(IO)}/T_{gEM(VS)}$ of 1.45 for DATA 10 and G_L, 1.48 for DATA 20 and G_H, 1.81 for DATA 10 and G_L, and 2.10 for DATA 10 and G_H. It can be seen that the difference of time complexity increases with the size of the sequences and especially with the level of ambiguity of the grammar.

The experimental results on memory complexity gave a ratio $M_{gEM(IO)}/M_{gEM(VS)}$ of 96.9144 for DATA 10 and G_L, 96.9168 for DATA 20 and G_H, 96.9198 for DATA 10 and G_L, and 96.9218 for DATA 10 and G_H. The size of the sequences and the level of ambiguity of the grammar has very little impact on the memory complexity.

Fig. 33 and 34 show the results for the two grammars with different levels of ambiguity after training with different training dataset sizes and the two algorithms, gEM(IO) and gEM(VS).

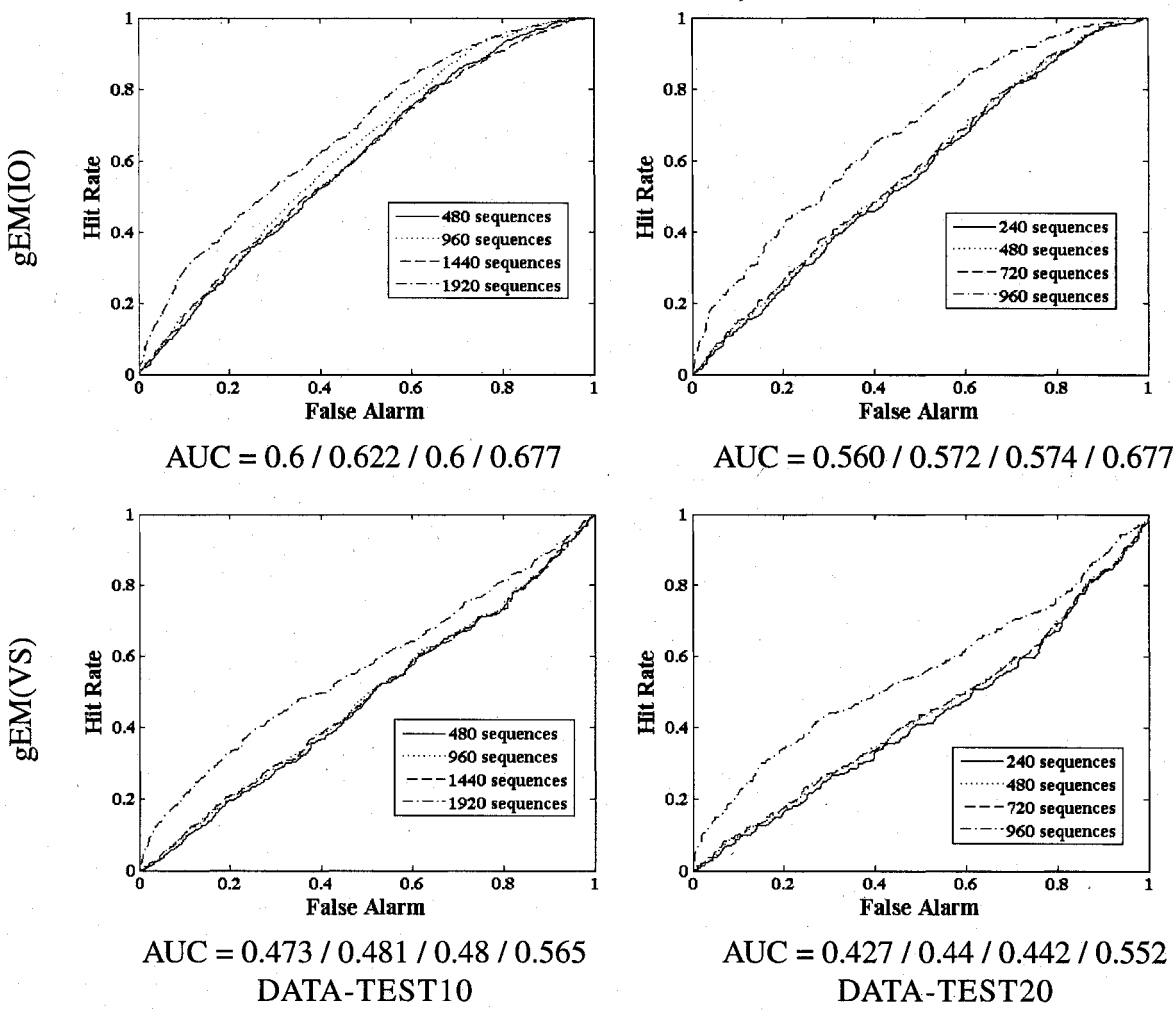


Figure 33 ROC curves for the low-ambiguity grammar after training with gEM(IO) and gEM(VS) for DATA-TEST10 and DATA-TEST20

It can be seen in these curves that gEM(VS) gives results similar to gEM(IO) when trying to differentiate species using DNA sequences. However, it appears clearly that the low-ambiguity grammar does not allow differentiating species from their DNA sequences. The high-ambiguity grammar gives better results, provided that enough data was used

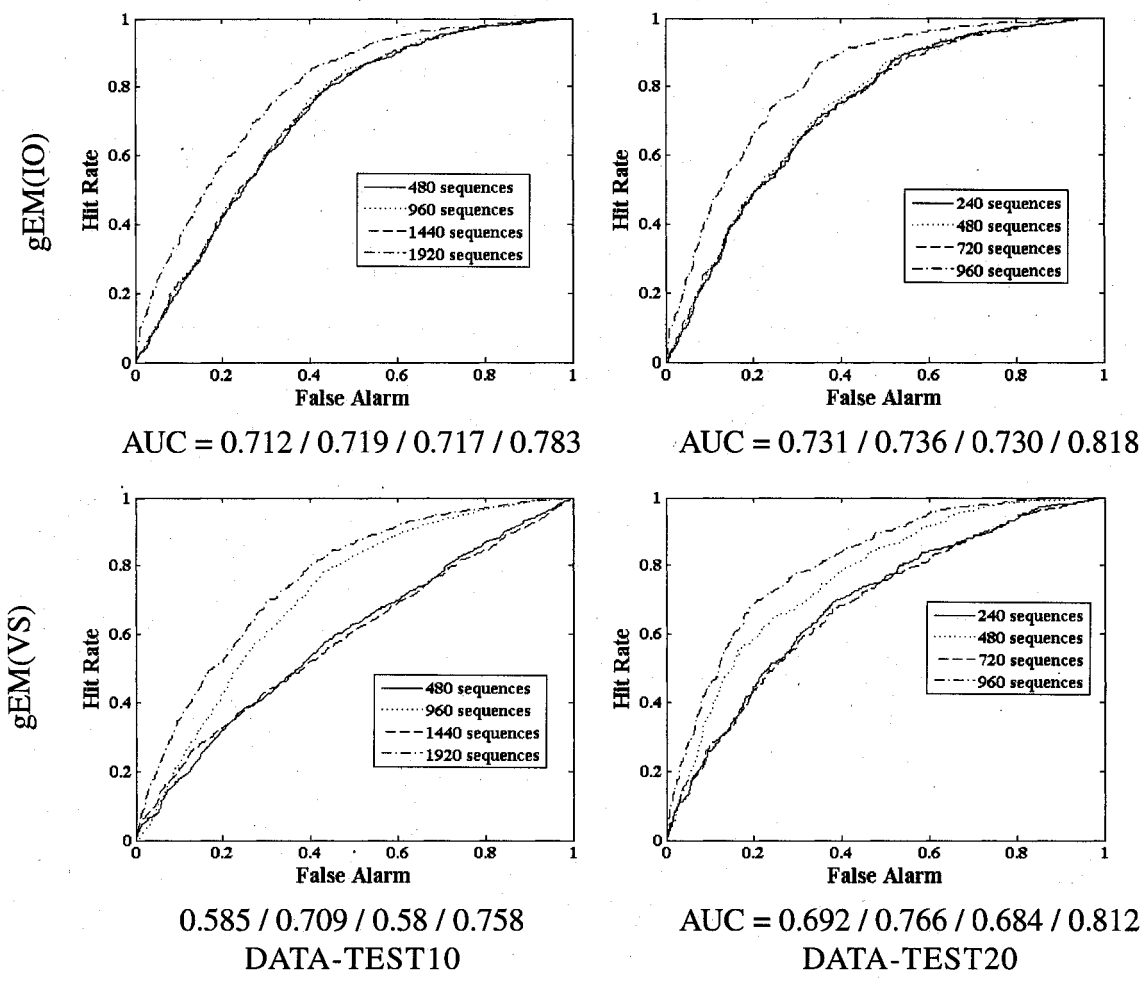


Figure 34 ROC curves for the high-ambiguity grammar after training with gEM(IO) and gEM(VS) for DATA-TEST10 and DATA-TEST20

for training. However these results are not good enough to be used in real applications. The problem may come from the fact that DNA sequences are very complex, and that the grammars that were used in this work was probably not complex enough to model it (it was originally design to get a good balance between ambiguity and time complexity). The two previous grammars only allow one-level dependency considering the pairs of nucleotides. In other words, the probability of emitting a pair of nucleotides only depends on the previous one. Another more complex grammar that allows two-levels dependency –

the probability of emitting a quadruplet of nucleotides depends on the previous one – was then designed as follow.

$$Start \mid Z1 \mid Z2 \mid Z3 \mid Z4 \mid Z5$$

$$\rightarrow SZ1\ S1 \mid SZ2\ S2 \mid SZ3\ S3 \mid SZ4\ S4 \mid SZ5\ S5 \mid Z6\ Z6 \mid S\ S \mid a \mid g \mid c \mid t$$

$$Z6 \rightarrow SZ1\ S1 \mid SZ2\ S2 \mid SZ3\ S3 \mid SZ4\ S4 \mid SZ5\ S5 \mid S\ S \mid a \mid g \mid c \mid t$$

$$SZ1 \rightarrow S1\ Z1 \mid S2\ Z1 \mid S3\ Z1 \mid S4\ Z1 \mid S5\ Z1$$

$$SZ2 \rightarrow S1\ Z2 \mid S2\ Z2 \mid S3\ Z2 \mid S4\ Z2 \mid S5\ Z2$$

$$SZ3 \rightarrow S1\ Z3 \mid S2\ Z3 \mid S3\ Z3 \mid S4\ Z3 \mid S5\ Z3$$

$$SZ4 \rightarrow S1\ Z4 \mid S2\ Z4 \mid S3\ Z4 \mid S4\ Z4 \mid S5\ Z4$$

$$SZ5 \rightarrow S1\ Z5 \mid S2\ Z5 \mid S3\ Z5 \mid S4\ Z5 \mid S5\ Z5$$

$$S \rightarrow A\,G \mid A\,C \mid A\,T \mid G\,A \mid G\,C \mid G\,T \mid C\,A \mid C\,G \mid C\,T \mid T\,A \mid T\,G \mid T\,C \mid a \mid g \mid c \mid t$$

$$S1 \rightarrow A\,A \mid A\,G \mid A\,C \mid A\,T$$

$$S2 \rightarrow G\,A \mid G\,G \mid G\,C \mid G\,T$$

$$S3 \rightarrow C\,A \mid C\,G \mid C\,C \mid C\,T$$

$$S4 \rightarrow T\,A \mid T\,G \mid T\,C \mid T\,T$$

$$A \rightarrow a$$

$$G \rightarrow g$$

$$C \rightarrow c$$

$$T \rightarrow t$$

A test on DATA-TEST10 after training using 1920 subsequences of nucleotides gives the result presented on Fig. 35. It can be seen that even if the result is not totally satisfying, it is better than for the previous grammars. So it seems that it is possible of creating more complex grammars that can model correctly DNA sequences, making so possible the differentiation of species using stochastic grammars trained with either gEM(IO) or gEM(VS) to model their DNA sequences.

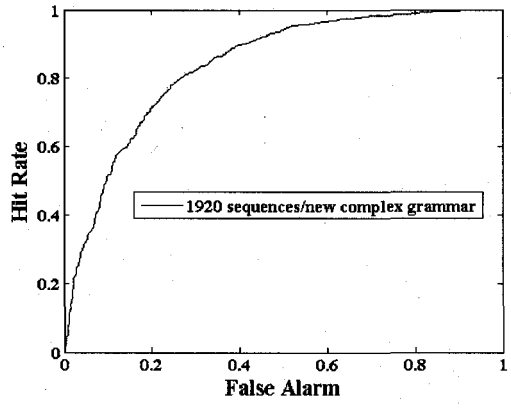


Figure 35 ROC curve for the new complex grammar after training with gEM(IO) for DATA-TEST10.

5.3 Discussion and Conclusions

Results indicate that gEM(VS) provides significantly lower convergence time and time complexity, at the expense of a moderately higher perplexity and insignificant difference of space complexity. It can be noted that gEM(VS) gives better results in terms of approximate perplexity, which may be more suitable when the structure of the best parse tree corresponding to a sequence is being studied.

These results confirm the differences that were expected between gEM(IO) and gEM(VS). Experiments on MFR data had already shown that gEM(VS) converges faster for ambiguous grammars, but no difference could be observed concerning the perplexity and the approximate perplexity, because of the MFR's language predictability and the associated low-ambiguity grammars.

BIBLIOGRAPHY

Amaya, F., Sanchez, J. A., and Benedi, J. M. (1999). Learning of Stochastic Context Free Grammars from Bracketed Corpora by Means of Reestimation Algorithms. *VIII Symposium on Pattern Recognition and Image Analysis*, volume 1, pages 119–126, Bilbao.

Anderberg, M. (1973). *Cluster Analysis for Applications*. Academic Press.

Aycock, J. and Horspool, R. N. (2002). Practical Earley Parsing. *The Computer Journal*, 45(6):620–630.

Baker, J. K. (1979). Trainable Grammars for Speech Recognition. *Speech Communications Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, MIT, Cambridge,Massachusset.

Baldi, P. and Chauvin, Y. (1994). Smooth On-line Learning Algorithms for Hidden Markov Models. *Neural Computation*, 6(2):307 – 318.

Baum, L. E. (1972). An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. *Inequalities*, 3:1–8.

Casacuberta, F. (1972). Statistical Estimation of Stochastic Context Free Grammars. *Pattern Recognition Letters*, 16:565–573.

Chaudhuri, R., Pham, S., and Garcia, O. (1983). Solution of an Open Problem on Probabilistic Grammars. *IEEE Trans. on Computers*, 32:748–750.

Chen, J. M. and Chaudhari, N. S. (2003). Improvement of the Inside-Outside Algorithm Using Prediction and Application to RNA Modelling. *Proc. of the Second Int. Conf. on machine Learning and Cybernetics*, pages 2255–2260, Xi'an.

Chung, P. J. and Bohme, J. F. (2003). Recursive EM Algorithm with Adaptive Step Size. *eventh Int. Symp. on Signal Processing and Its Applications, 2003*, volume 2, pages 519–522, Paris, France.

Corman, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.

Crocker, M. (2003). Probabilistic Context-Free Grammars: Inside and Outside Probabilities, and Viterbi Parsing. Mathematische Grundlagen III: Statistische Methoden, Lecture Notes.

Davies, C. L. and Hollands, P. (1982). Automatic Processing for ESM. *IEE Proc. F: Communications, Radar and Signal Processing*, pages 164–171.

Digalakis, V. V. (1999). Online Adaptation of Hidden Markov Models Using Incremental Estimation Algorithms. *IEEE Transactions on Speech and Audio Processing*, volume 7 of *3*, pages 253–261.

Dilkes, F. A. (2004a). Pulse Processing Techniques for Stochastic Regular Grammar Models in Electronic Support. unpublished Technical report, DRDC Ottawa.

Dilkes, F. A. (2004b). Statistical Cross-Correlation Algorithms for Temporal Pulse Profiles. Technical report TM 2004-220, DRDC-Ottawa.

Dilkes, F. A. (2005a). Linguistic Radar Modeling for Electronic Support: Pulse Processing for Stochastic Regular Grammars. *Proc. of 2004 Workshop on Defence Applications of Signal Processing*, Midway, Utah.

Dilkes, F. A. (2005b). Statistical Decision Threshold for Pulse Template Cross-Correlators. Technical report DRDC-OTTAWA-TM-2005-167, DRDC-Ottawa.

Dowell, R. and Eddy, S. (2004). Evaluation of Several Lightweight Stochastic Context-Free Grammars for RNA Secondary Structure Prediction. *BMC Bioinformatics*, 5(1):71.

Earley, J. (1970). An Efficient Context-Free Parsing Algorithm. *Communication of the ACM*, volume 13 of *2*, pages 84–102.

Elton, S. D. (2001). A Cross-Correlation Technique for Radar Emitter Recognition Using Pulse Time of Arrival Information. *Defence Applications in Signal Processing*, pages 19–23. Defence Science and Technology Organisation, Australia.

Estève, Y. (2002). *Intégration de Sources de Connaissances pour la Modélisation Stochastique du Langage Appliquée à la Parole Continue dans un Contexte de Dialogue Oral Homme-Machine*. PhD thesis, Université d'Avignon et des pays de Vaucluse.

Fan, F. (2004). Review on RNA Sequence Element Analysis. http://www.micab.umn.edu /8006/litreviews/review-RNA.pdf.

Fawcett, T. (2003). Roc Graphs: Notes and Practical Considerations for Data Mining Researchers. Technical report HPL-2003-4, HP Labs.

Fu, K. S. (1982). *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, N.J.: Prentice-Hall.

Fujisaki, T., Jelinek, F., Cocke, J., Black, E., and Nishino, T. (1989). A Probabilistic Parsing Method for Sentence Disambiguation. *International Workshop on Parsing Technologies*, pages 85–94, Pittsburgh, PA.

Gotoh, Y., Hochberg, M. M., and Silverman, H. F. (1998). Efficient Training Algorithms for HMM's Using Incremental Estimation. *IEEE Transactions on Speech and Audio Processing*, 6:539–548.

Granger, E. (2002). *Une Étude des Réseaux de Neurones Artificiels pour la Classification Rapide d'Impulsions Radars.* PhD thesis, Université de Montréal, http://www.livia.etsmtl.ca/publications/2002/Granger_PhD_2002.pdf.

Haykin, S. and Currie, B. (2003). Automatic Radar Emitter Recognition Using Hidden Markov Models. unpublished Technical report, DRDC Ottawa.

Heeringa, B. and Oates, T. (2001a). Incrementally Learning the Parameters of Stochastic Context-Free Grammars Using Summary Statistics and Repeated Sampling. University of Massachusetts Amherst, unpublished.

Heeringa, B. and Oates, T. (2001b). Two Algorithms for Learning Parameters of Stochastic Context-Free Grammars. *Working Notes of the 2001 AAAI Fall Symposium on Using Uncertainty within Computation*, pages 58–62.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 2nd edition.

Hori, C., Katoh, M., Ito, A., and Kohda, M. (2002). Construction and Evaluation of Language Models Based on Stochastic Context-Free Grammars for Speech Recognition. *Systems and Computers in Japan*, 33:48–59.

Ito, A., Hori, C., Katoh, M., and Kohda, M. (2001). Language Modeling by Stochastic Dependency Grammar for Japanese Speech Recognition. *Systems and Computers in Japan*, 32:10–15.

Jauvin, C. (2003). *Quelques Modèles de Langage Statistiques et Graphiques Lissés avec WordNet.* PhD thesis, Université de Montréal.

Jorgensen, M. (1999). A Dynamic EM Algorithm for Estimating Mixture Proportions. *Statistics and Computing*, 9:299–302.

Jurafsky, D., Wooters, C., Segal, J., Stolcke, A., Fosler, E., Tajchaman, G., and Morgan, N. (1995). Using a Stochastic Context-Free Grammar as a Language Model for Speech Recognition. *Proc. of the 1995 Int. Conf. on Acoustics, Speech, and Signal Processing*, volume 1, pages 189–192, Detroit, Michigan.

Kato, Y., Matsubara, S., Toyama, K., and Inagaki, Y. (2005). Incremental dependency parsing based on headed context-free grammar. *Systems and Computers in Japan*, 36(2):63–77.

Kupiec, J. (1992). Hidden Markov Estimation for Unrestricted Stochastic Context-Free Grammars. *Proc. of the IEEE Conf. on Acoustics, Speech and Signal*, volume 1, pages 177–180, San Francisco, California.

Lari, K. and Young, S. (1990). The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language*, 4:35–56.

Lari, K. and Young, S. (1991). The Application of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language*, 5:237–257.

Latombe, G., Granger, E., and Dilkes, F. A. (2006a). Fast Incremental Learning of Grammatical Probabilities in Radar Electronic Support. *IEEE Trans. Aerospace and Electronic Systems*, under review.

Latombe, G., Granger, E., and Dilkes, F. A. (2006b). Fast Incremental Learning of Grammatical Probabilities in Radar Electronic Support. Technical report, Defence R&D Canada - Ottawa, under review.

Latombe, G., Granger, E., and Dilkes, F. A. (2006c). Fast Incremental Techniques for Learning Production Rule Probabilities in Radar Electronic Support. *in Proc. of Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 14–19, Toulouse, France.

Latombe, G., Granger, E., and Dilkes, F. A. (2006d). Incremental Learning of Stochastic Grammars with Graphical EM in Radar Electronic Support. in Proc. of Int. Conf. on Acoustics, Speech, and Signal Processing, under review.

Latombe, G., Granger, E., and Dilkes, F. A. (2006e). Maximum Likelihood and the Graphical EM Algorithm. *in Proc. of Canadian Conf. on Electrical and Computer Engineering*, Ottawa, Ontario, under press.

Latombe, G., Granger, E., and Dilkes, F. A. (2006f). Incremental Versions of Graphical EM for Fast Learning of Grammatical Probabilities in Radar Electronic Support. *Signal Processing*, in preparation.

Lavoie, P. (2001). Hidden Markov Modeling for Radar Electronic Warfare. Technical report DREO-TM-2001-127, Defence R&D Canada - Ottawa.

Lucke, H. (1994). Reducing the Computational Complexity for Inferring Stochastic Context-Free Grammar Rules from Example Text. *Proc. of Int. Conf. on Acoustics, Speech, and Signal Processing*, volume 1, pages 353–356, Adelaide, Australia.

Nakamura, K. and Matsumoto, M. (2002). *Incremental Learning of Context-Free Grammars*, chapter Grammatical Inference: Algorithms and Applications (ICGI); Amsterdam, the Netherlands, page 174. Berlin Heidelberg, Germany: Springer-Verlag.

Neal, R. M. and Hinton, G. E. (1998). *A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants*, chapter Learning in graphical models, pages 355–368. Cambridge, MA: MIT Press.

Nevado, F., Sanchez, J. A., and Benedi, J. M. (2000). Combination of Estimation Algorithms and Grammatical Inference Techniques to Learn Stochastic Context-Free Grammars. *Proc. of 5th International Colloquium, ICGI 2000*, pages 164–171, Lisbon, Portugal.

Ney, H. (1992). *Stochastic Grammars and Pattern Recognition*, chapter Speech Recognition and Understanding: Recent Advances, pages 319–344. Laface, P. and Mori, R. New York: Springer-Verlag.

Nijholt, A. (1991). The CYK-Approach to Serial and Parallel Parsing. *Quarterly Journal of Linguistics of the Language Research Institute of Seoul National University*, Language Research 27(2):229–254.

Oates, T. and Heeringa, B. (2002). Estimating Grammar Parameters Using Bounded Memory. P. Adriaans, H. F. and van Zaanen, M., editors, *Proceedings of the Sixth International Colloquium on Grammatical Inference*, pages 185–198.

Osborne, M. and Briscoe, T. (1997). *Learning Stochastic Categorical Grammars*, chapter Speech Recognition and Understanding: Recent Advances, pages 80–87. Ellison, T. M.

Polikar, R., Upda, L., Upda, S., and Honavar, V. (2001). Learn++: an Incremental Learning Algorithm for Supervised Neural Networks. *IEEE Trans. Syst. Man Cybern. C, Appl. Rev. (USA)*, 31(4):497–508.

Ra, D.-Y. and Stockman, G. C. (1999). A New One-Pass Algorithm for Estimating Stochastic Context-Free Grammars. *Information Processing Letters*, 72:37–45.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. IEEE*, volume 77 of 2, pages 257–286.

Rogers, J. A. V. (1985). ESM processor system for high pulse density radar environments. *IEE Proceedings F: Communications Radar and Signal Processing*, 132:621–625.

Sakakibara, Y. (1990). Learning Context Free Grammars from Structural Data in Polynomial Time. *Theoretical Computer Science*, 76:223–242.

Sakakibara, Y., Brown, M., Mian, I. S., Hughey, R., Underwood, R. C., and Haussler, D. (1993). Stochastic Context-Free Grammars for tRNA Modeling. Technical report UCSC-CRL-93-16, University of California at Santa Cruz TR.

Sakakibara, Y., Brown, M., Underwood, R. C., Mian, I. S., and Haussler, D. (1994). Stochastic Context-Free Grammars for Modeling tRNA. *Nucleic Acid Research*, 22:5112–5120.

Sanchez, J. A. and Benedi, J. M. (1997). Consistency of Stochastic Context-Free Grammars from Probabilistic Estimation Based on Growth Transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:1052–1055.

Sanchez, J. A. and Benedi, J. M. (1999a). Learning of Stochastic Context-Free by Means of Estimation Algorithm. *EUROSPEECH'99*, pages 1799–1802, Budapest, Hungary.

Sanchez, J. A. and Benedi, J. M. (1999b). Probabilistic Estimation of Stochastic Context-Free Grammars from the K-Best Derivations. *VIII Symposium on Pattern Recognition and Image Analysis*, volume 2, pages 7–14, Bilbao, Spain.

Sanchez, J. A., Benedi, J. M., and Casacuberta, F. (1996). Comparison Between the Inside-Outside Algorithm and the Viterbi Algorithm for Stochastic Context-Free Grammars. Verlag, S., editor, *Lecture notes in AI*, volume 1121, pages 50–59.

Sato, M. and Ishii, S. (2000). On-line EM Algorithm for the Normalized Gaussian Network. *Neural Computation*, 12(2):407–432.

Sato, T., Abe, S., Kameya, Y., and Shirai, K. (2001). A Separate-and-Learn Approach to EM Learning of PCFGs. *Proc. of the 6th Natural Language Processing Pacific Rim Symposium*, pages 255–262.

Sato, T. and Kameya, Y. (2001). Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *Journal of Artificial Intelligence Research*, 2:391–454.

Scott, C. and Nowak, R. (2004). Sufficient Statistics. Connexion Web site, http://cnx.rice.edu/content/m11481/1.7/.

Stolcke, A. (1995). An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities. *Computational Linguistics*, 21:2:165–201.

Titterington, D. M. (1984). Recursive Parameters Estimation using Icomplete Data. *J. Royal Statistic Soc.*, 46:257–267.

Underwood, R. C. (1994). Stochastic Context-Free Grammars for Modeling Three Spliceosomal Small Nuclear Ribonucleic Acids. Technical report UCSC-CRL-94-23, University of California at Santa Cruz TR.

Uratani, N., Takezawa, T., Matsuo, H., and Morita, C. (1994). ATR Integrated Speech and Language Database. Technical report TR-IT-0056, Telecommunications Research Laboratories.

Visnevski, N. (2005). *Syntactic Modeling of Multi-Function Radars.* PhD thesis, McMaster University, http://soma.ece. mcmaster.ca/~visnev/Resources/PHDThesis/.

Visnevski, N., Dilkes, F. A., Haykin, S., Krisnamurthy, V., and Currie, B. (2005). Non-Self-Embedding Context-Free Grammars for Multi-Function Radar Modeling - Electronic Warfare Application. *Proc. of the 2005 IEEE International Radar Conference*, Arlington, Virginia.

Visnevski, N., Krisnamurthy, V., Haykin, S., Currie, B., Dilkes, F., , and Lavoie, P. (2003). Multi-Function Radar Emitter Modeling: a Stochastic Discrete Event System Approach. *Proc. of the 42nd IEEE Conf. on Decision and Control (CDC)*, Maui, Hawaii.

Wiley, R. G. (1993). *Electronic Intelligence: the Analysis of Radar Signals, 2nd ed.* Norwood, MA:Artech House.