Applied Soft Computing xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support

Guillaume Latombe[a], Eric Granger[a,*], Fred A. Dilkes[b]

[a] Laboratoire d'imagerie, de vision et d'intelligence artificielle, École de technologie supérieure, Montreal, Canada
[b] Defence R&D Canada – Ottawa, Department of National Defence, Ottawa, Canada

ARTICLE INFO

Article history:
Received 29 March 2011
Received in revised form
24 November 2011
Accepted 17 February 2012
Available online xxx

Keywords:
Syntactic pattern recognition
Stochastic Context-Free Grammars
On-line learning
Incremental learning
Graphical Expectation-Maximization
HOLA
Radar Electronic Support

ABSTRACT

Although Stochastic Context-Free Grammars (SCFGs) appear promising for the recognition and threat assessment of complex radar emitters in radar Electronic Support (ES) systems, techniques for learning their production rule probabilities are computationally demanding, and cannot efficiently reflect changes in operational environments. On-line learning techniques are needed in ES applications to adapt SCFG probabilities rapidly, as new training data are collected from the field. In this paper, an efficient on-line version of the fast learning technique known as the graphical Expectation-Maximization (gEM) technique – called on-line gEM (ogEM) – is proposed. A second technique called on-line gEM with discount factor (ogEM-df) expands ogEM to allow for tuning the learning rate. The performance of these new techniques has been compared to HOLA, the only other fast on-line learning technique, from several perspectives – perplexity, error rate, complexity, and convergence time – and using complex radar signals. The impact on performance of factors like the size of new data blocks, and the level of ambiguity of grammars has been observed. Results indicate that on-line learning of new training data blocks with ogEMand ogEM-df provides the same level of accuracy as batch learning with gEM using all cumulative data from the start, even for small data blocks. As expected, on-line learning significantly reduces the overall time and memory complexities associated with updating probabilities with new data in an operational environment. Finally, while the computational complexity and memory requirements of ogEM and ogEM-df may be greater than that of HOLA, they both provide a significantly higher level of accuracy.

1. Introduction

Electronic Support (ES) involves the passive interception, detection and analysis of radiated electromagnetic signals in the context of military surveillance. It provides valuable information for real-time situation awareness, threat assessment, and timely deployment of counter-measures [1,2]. Two critical functions of radar ES are the recognition of radar emitters from intercepted signals, and the estimation of the instantaneous level of threat that they pose. The recent proliferation and complexity of 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, such as carrier frequency, pulse repetition frequency, and pulse width. With the advent of automatic electronic switching designed to optimize radar performance, modern radars, and especially multifunction radars (MFRs), are often too complex to be recognized in this way. MFRs will continuously and autonomously change their transmitted signals in response to various events in their dynamically changing environment. In order to exploit the dynamic nature of modern radar systems, Stochastic Context Free Grammars (SCFGs) [3] have been proposed for modeling MFR signals [4,5], resulting in compact formal representations that can be exploited for recognition of radar emitters, and for estimation of their instantaneous level of threat.

With the advent of automatic electronic switching designed to optimize radar performance, modern radars, and especially multifunction radars (MFRs), often have complex functioning, and will continuously and autonomously change their transmitted signals in response to various events in their dynamically changing environment. Stochastic Context Free Grammars (SCFGs) [1] have been proposed for modeling MFR signals [2,3], resulting in compact formal representations that can be exploited for recognition of radar emitters, and for estimation of their instantaneous level of threat.

Given a set of training sequences collected in the field, one challenge to the practical application of SCFGs is the task of estimating or learning probability distributions associated

* Corresponding author at: École de technologie supérieure, 1100 rue Notre-Dame Ouest, Montreal, Quebec H3C 1K3, Canada. Tel.: +1 514 396 8650.
E-mail address: eric.granger@etsmtl.ca (E. Granger).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2012.02.022

with their production rules. Their estimation is typically performed based on Maximum Likelihood (ML) optimization using an Expectation-Maximization (EM) technique. The most popular EM-based techniques are the Inside–Outside (IO) [4] and the Viterbi Score (VS) [5] algorithms. 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. Computing the probability, for each iteration, that a SCFG with M_{nt} non-terminal symbols will generate a given string of length L has a time complexity of $O(M_{nt}^3 \cdot L^3)$ and a memory complexity of $O(M_{nt} \cdot L^2)$. Moreover, neither one of these techniques allows for on-line learning of new data that may become available during operations.

In radar ES applications, new data from an operational environment or other sources often becomes available at different points in time. This data may be incorporated into blocks of training sequences collected in the field at those times. The ability to efficiently adapt SCFG probabilities as new data become available, through *on-line learning*, is therefore an undisputed asset for accurate recognition of radar emitters. With on-line learning, training sequences are not accumulated and stored in memory, and SCFGs probabilities are not learned from the start on all accumulated data. Therefore, it can considerably reduce the memory requirements and computational complexity needed to update a SCFG.

Several alternatives to IO and VS have been proposed to accelerate learning of SCFG probabilities in different application domains [6]. A promising approach for fast learning involves pre-computing data structures, using tools like the Earley [7] or CYK [8,9] chart parsers during the pre-processing phase, and then exploiting these data structures to accelerate the iterative probability re-estimation process [6,10–13]. In practice, all techniques based on this approach compute exactly the same values as IO or VS during the iterative process, yet may give lower time complexity per iteration, at the expense of an increased memory complexity.

One of the more recent techniques to adopt this approach is the graphical Expectation-Maximization (gEM) algorithm [12]. During pre-processing, a chart parser produces 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, a variant of the IO-like algorithm is applied, based on these support graphs. The time complexity per iteration and memory complexity is $O(M_{nt}^3 \cdot L^3)$. In contrast, the technique called HOLA [14] is based on summary statistics, and on parsing that does not directly optimize the likelihood of a training data. Instead, it optimizes the relative entropy between the distribution of the rules obtained after having parsed the training data, and the distribution of the rules obtained after having parsed a data generated by the grammar. At each iteration, a new data is generated by the grammar and compared with the training set until the sample distribution converges to the training distribution. It has the advantage of having a very low time complexity per iteration and memory complexity of $O(M_{nt}^3)$, that is independent of L.

Results from a comparative study with radar data [6] indicate that gEM systematically provides a higher level of accuracy than HOLA. However, the execution time and memory requirements of HOLA are orders of magnitude lower than that of gEM, since the computational complexity of HOLA is bound by the number of SCFG rules, not by the amount of training data. Furthermore, HOLA can perform on-line learning – it can learn SCFG probabilities from new data without accessing previous data. Given a new block of training data, the summary statistics are updated, and learning may resume using the SCFG probabilities produced from previous training data. With the exception of HOLA, the iterative re-estimation process of all the algorithm already cited must start from the beginning, on all cumulative data, to account for new training data. The result from the pre-processing is stored and remains unchanged during the iterative process if new data is added, yet the result from the iterative process is rendered obsolete. In light of these results, an on-line version of gEM, that can efficiently reflect the changes in an operational environment, would be very desirable for radar ES applications.

In this paper, two novel derivations of gEM are proposed for on-line learning. The first one, called on-line gEM (ogEM), allows for efficient on-line learning based on sufficient statistics, whereas the second one, called on-line gEM with discount factor (ogEM-df), expands on ogEM by allowing to tuning the learning rate during on-line learning. Both algorithms are compared to HOLA. 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 (the likelihood of a test data set), classification rate over radar states, convergence time, and time and memory complexity. The data sets used in our proof-of-concept computer simulations describe electromagnetic signals transmitted from different MFR systems. An experimental protocol has been defined such that the impact on performance of factors like the size of new data blocks for on-line learning, and the level of ambiguity of grammars, may be observed. Other relevant areas of application for ogEM and ogEM-df include bio-informatics, video content analysis, and character and speech recognition.

The rest of this paper is structured into five sections as follows. The next section provides some background information on grammatical modeling in the context of radar ES applications. Section 3 presents the approach behind EM-based learning of SCFG probabilities, as well as the original gEM technique. The main features of ogEM and ogEM-df are outlined in Section 4, and the experimental methodology used to compare these techniques (protocol for computer simulations, radar datasets, performance measures) is described in Section 5. Finally, the results of computer simulations and complexity estimates are presented in Section 6. The advantages and drawbacks of these techniques for learning SCFG probabilities on radar data are discussed with ES application in mind.

2. Grammatical modeling for radar signal recognition in Electronic Support

In modern pulsed radar applications, a signal may be generated by a multifunction radar (MFR) in response to its operating environment, and pulse patterns, also called *words* hereafter, can be identified in order to form a sequence. The algorithm controlling the function of a MFR is typically designed as an automaton whose state transitions are driven by the stochastic behavior of the environment, target returns and operator controls [15]. Since radar systems are expected to behave in a causal fashion with finite memory, it should satisfy the formal requirements of a regular language [8] and admit a syntactic description in terms of a finite-state machine. However, some radar systems exhibit a signal behavior that requires a large number of states and sub-states [15].

Alternative syntactic models that are not based on finite-state machines offer the analyst more flexibility to include and exclude different arrangements of operational tasking without resorting to a proliferation of model parameters. One simple extension of the finite-state model, known as a *Context-Free Grammar* (CFG) [8], naturally permits a sequential arrangement of operational tasks. Mathematically, a CFG is a quadruplet $G = (V, N, R, Start)$ consisting of a finite set V of *terminal symbols* known as a vocabulary (which will consists in the set of words possibly emitted by the MFR), a finite set N of *non-terminals symbols* including the states of the MFR, and satisfying $V \cap N = \emptyset$, a set R of production rules, and a unique starting non-terminal symbol $Start \in N$. Each production rule in R has the

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

form $A \to \lambda$, where $A \in N$ is a single non-terminal symbol and λ is a finite sequence of symbols drawn from $N \cup V$.

Each CFG G can be used to define a deterministic formal language $Lg(G)$ using a graphical construction. A *parse tree* [8] is a labeled, directed, rooted tree graph. Each parent node must be labeled by a non-terminal symbol from N and each child node is labeled by a symbol from either N or V. If a parent node is labeled by $A \in N$ and its k children are labeled by $X_1, X_2, \ldots, X_k \in N \cup V$, then the production rule $A \to X_1 X_2, \ldots, X_k$ must appear in R. Any sequence of terminal symbols that is yielded by a parse tree rooted at the *Start* symbol is in the language $Lg(G)$.

In practice, the syntactic radar models must be designed by an ES domain analyst with sufficient prior intelligence to construct grammatical production rules for threat systems of interest. 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. Such grammars are called *Stochastic Context-Free Grammars* (SCFGs) [1]. SCFGs form an important class of grammars that is widely used to characterize the probabilistic modeling of language in computational linguistic and automatic speech recognition and understanding [1], or in RNA secondary structure prediction [16].

A SCFG is defined as a pair of the form $G_s = (G, \Theta)$ where G is a CFG and Θ is a vector containing probability distributions for each non-terminal symbol. The components of Θ are simply denoted by $\theta(A \to \lambda)$ which represents the *probability of the production rule* $A \to \lambda \in R$. These define distributions for each $A \in N$ in the sense that $0 \le \theta(A \to \lambda) \le 1$ for $\forall \lambda$, and $\sum_\lambda \theta(A \to \lambda) = 1$. The probability of one parse tree d_x of the sequence x of terminal symbols is defined as:

$$P(x, d_x|G_s) = \prod_A \prod_\lambda \theta(A \to \lambda)^{N(A \to \lambda, d_x)} \tag{1}$$

where $N(A \to \lambda, d_x)$ is the frequency of appearance the rule $A \to \lambda$ in d_x. The *probability of the sequence* x with respect to Δ_x, a set of possible parse trees d_x leading to the sequence x, is defined as:

$$P(x, \Delta_x|G_s) = \sum_{d_x \in \Delta_x} P(x, d_x|G_s) \tag{2}$$

and the *probability of the best parse tree of the sequence* x from a set of parse trees Δ_x is defined as:

$$\widehat{P}(x|G_s) = \max_{d_x} P(x, d_x|G_s) \tag{3}$$

Finally, the *best parse tree*, $\widehat{d}_x = \text{argmax}_{d_x} P(x, d_x|G_s)$, is defined as the argument that maximizes Eq. (3).

3. Fast learning of SCFG probabilities with graphical EM

Most Expectation-Maximization (EM)-based techniques for learning the production rule probabilities of a SCFG are based on the classical Maximum Likelihood approach. Given a SCFG G_s, a joint likelihood of any finite collection of training sequences $\Omega \subset Lg(G_s)$ with repetitions allowed, and a specified set of parse trees Δ_x for each $x \in \Omega$, is given by [17]:

$$P(\Omega, \Delta_\Omega|G_s) = \prod_{x \in \Omega} P(x, \Delta_x|G_s) \tag{4}$$

where $P(x, \Delta_x|G_s)$ is the sequence probability given by Eq. (2). Here Δ_Ω represents the set of derivation trees considered to form the sequences in Ω. It can be noted that Eq. (4) coincides with the *likelihood* of the entire set Ω [17] when Δ_x is identified with the set $\overline{\Delta_x}$ consisting of every possible parse tree permitted by G_s rooted at *Start* and yielding $x \in \Omega$. It coincides with the likelihood of the best parses when Δ_x contains only the single most probable parse tree $\widehat{d}_x$ determined by Eq. (3) for each $x \in \Omega$. An alternative possibility [17] would be for Δ_x to represent the k most probable parse trees for each sequence in the training set, for $k > 1$.

The *Maximum Likelihood* (ML) approach for learning the probabilities of the grammar consists of maximizing the likelihood Eq. (4) with respect to the production rule probabilities $\theta(A \to \lambda)$. Optimization of Eq. (4) is typically implemented using an iterative EM re-estimation technique. At each iteration, the following function can be applied to re-estimate the production rule probabilities to approach a local maximum of Eq. (4) [17]:

$$\theta'(A \to \lambda) = \frac{\eta(A \to \lambda)}{\sum_\mu \eta(A \to \mu)}. \tag{5a}$$

where

$$\eta(A \to \lambda) = \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x|G_s)} \sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x|G_s) \tag{5b}$$

is known as a *balanced frequency* of the production rule $A \to \lambda \in R$ with respect to the training set Ω. For any non-terminal $A \in N$ such that $\sum_\mu \eta(A \to \mu) = 0$, production rule is not updated so that Eq. (5a) is simply replaced by $\theta'(A \to \lambda) = \theta(A \to \lambda)$.

The most well-known EM-based techniques is the Inside–Outside (IO) [4], which seeks to maximize the overall likelihood Eq. (4) of a training dataset, and proceeds as follows. For a SCFG with specified production rule probabilities, and a sequence $(w_1, \ldots, w_L)$, define an *inside probability*

$$\alpha_{A(i,j)} = P(A \Rightarrow w_{i+1}, \ldots, w_j)$$

to account for all possible sub-trees rooted at each $A \in N$ and yielding each sub-sequence $(w_{i+1}, \ldots, w_j)$. One can also define a corresponding *outside probability*

$$\beta_{A(i,j)} = P(Start \Rightarrow w_1, \ldots, w_i A w_{j+1}, \ldots, w_L)$$

to account for all possible sub-trees rooted at *Start* and yielding the sequence $(w_1, \ldots, w_L)$ in which the non-terminal $A \in N$ takes the place of a specified sub-sequence,

The IO algorithm [4] proceeds as though all inside and outside probabilities are non-vanishing for any sequence x, while some relevant probabilities may necessarily vanish as dictated by the structure of the grammar. The graphical EM (gEM) technique proposed by Sato and Kameya [12] operates in a similar way to the IO algorithm. The main difference lies in the fact that, to re-estimate the probabilities, gEM evaluates only those probabilities $\alpha_{A(i,j)}$ and $\beta_{A(i,j)}$ for which the grammatical production rules permit the non-terminal A to appear at the root of a sub-tree yielding the subsequence $(w_{i+1}, \ldots, w_j)$. A set of data structures known as *support graphs* is constructed by means of a parser, such as the CYK or the Earley parser, to represent exactly those parse trees that may lead to each training sequence. Fig. 1 shows the support graphs created from the sequence (6, 6, 6, 6, 10), resulting from an MFR of type Mercury, and whose CNF context-free grammar is described in [6].

Once the support graphs have been pre-computed, each iteration of the gEM algorithm may be applied on the support graphs using the following four steps [12].

1. *Compute inside and explanation probabilities for each support graph:* Each inside probability $\alpha_{A(i,j)}$ is computed only if $A(i, j)$ is true, and therefore represents the root node of some support graph. For support graphs rooted at nodes of the form $A(j-1, j)$, corresponding to a sub-sequence containing one non-terminal (w_j), the inside probability is $\alpha_{A(j-1,j)} = \theta(A \to w_j)$. For longer sub-sequences, only those summation terms that correspond to

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

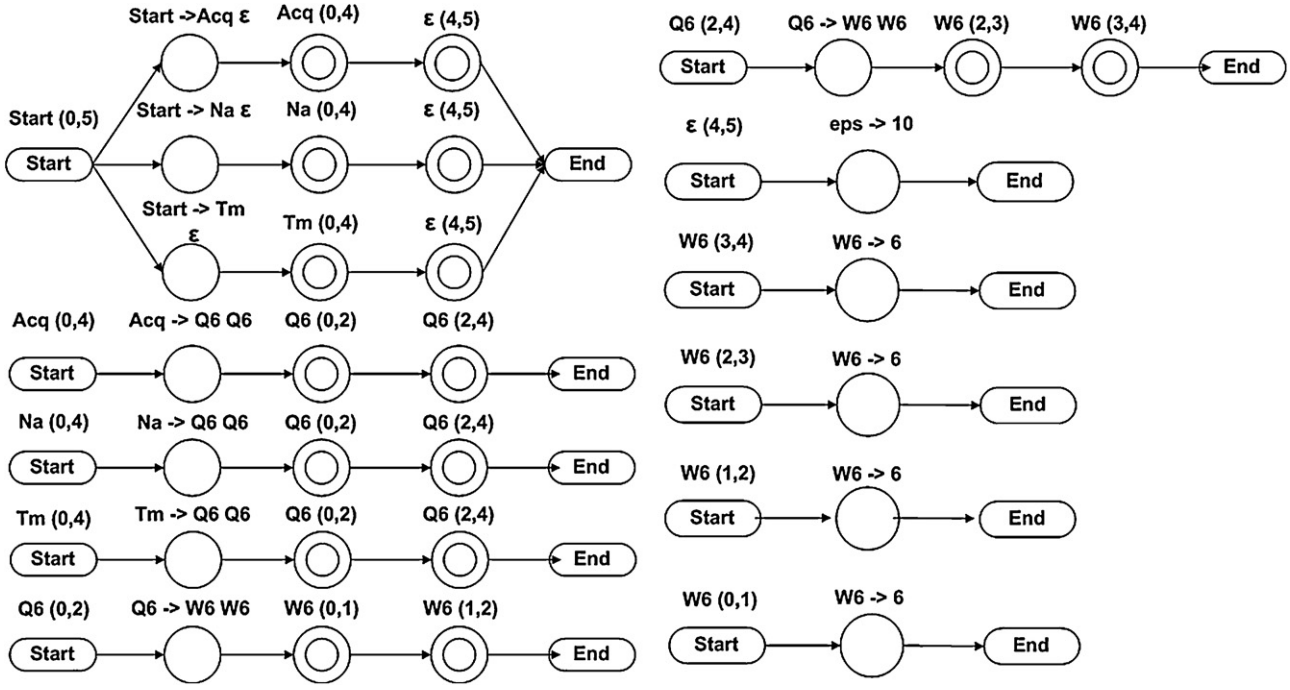


Fig. 1. Support graphs produced for the sequence (6, 6, 6, 6, 10), resulting from an MFR of type Mercury. In this graph, simple and concentric circles are used to distinguish between a production rule and the fact that a non-terminal produces a sub-sequence of words, respectively.

branches of the relevant support graph need to be considered. If $i < j-1$ and $A(i,j)$ is true, then an *explanation probability*

$$\alpha_{A(i,j)\to B(i,k)C(k,j)} = \theta(A \to BC)\alpha_{B(i,k)}\alpha_{C(k,j)}. \tag{6a}$$

is computed for each branch of the support graph of the form $A(i,j) \to B(i,k)C(k,j)$. The inside probability of the entire support graph rooted at $A(i,j)$ is then given by

$$\alpha_{A(i,j)} = \sum_{A(i,j)\to B(i,k)C(k,j)} \alpha_{A(i,j)\to B(i,k)C(k,j)} \tag{6b}$$

where the summation extends over all branches of the support graph. The recursive procedure proceeds in a bottom-up fashion with respect to the partial ordering since $\alpha_{B(i,k)}$ may contribute to $\alpha_{A(i,j)}$ only if the support graph rooted at $A(i,j)$ precedes that of $B(i,k)$.

2. *Compute outside probabilities:* Each outside probability $\beta_{A(i,j)}$ is computed only if the condition $A(i,j)$ is true, and consequently appears at the root of a support graph. The outside probability of the highest support graph is $\beta_{Start(0,L)} = 1$. The recursion relation for $\beta_{A(i,j)}$ has a straightforward interpretation in terms of support graphs since the two summations receive non-vanishing contributions only from combinations for which $B(i,k) \to A(i,j)C(j,k)$ and $B(k,j) \to C(k,i)A(i,j)$ are true respectively. Other combinations for which no such support graph branches exist will not contribute. The outside probability of the support graph rooted at $A(i,j)$ is may be written in terms of the explanation probabilities as follows:

$$\beta_{A(i,j)} = \sum_{B(i,k)\to A(i,j)C(j,k)} \frac{\beta_{B(i,k)}\alpha_{B(i,k)\to A(i,j)C(j,k)}}{\alpha_{A(i,j)}} + \sum_{B(k,j)\to C(k,i)A(i,j)} \frac{\beta_{B(k,j)}\alpha_{B(k,j)\to C(k,i)A(i,j)}}{\alpha_{A(i,j)}}. \tag{7a}$$

The summation in Eq. (7a) extends over every branch of every support graph in which $A(i,j)$ appears as specified. The summation can be efficiently implemented using a running tally for each $\beta_{A(i,j)}$ by sequentially examining each branch of each support graph and adding the summand to the appropriate outside probability. These equations are implemented using a top-down recursion since, for example, $\beta_{B(k,j)}$ contributes to $\beta_{A(i,j)}$ only if the support graph rooted at $B(k,j)$ precedes that of $A(i,j)$.

3. *Compute balanced frequencies of the rules:* The balanced frequencies are computed using

$$\eta(A \to a) = \sum_{x\in\Omega}\left(\sum_{A(j-1,j)\to w_j} \frac{\beta_{A(j-1,j)}\alpha_{A(j-1,j)}}{\alpha_{Start(0,L)}}\right)$$
$$\eta(A \to BC) = \sum_{x\in\Omega}\left(\sum_{A(i,j)\to B(i,k)C(k,j)} \frac{\beta_{A(i,j)}\alpha_{A(i,j)\to B(i,k)C(k,j)}}{\alpha_{Start(0,L)}}\right) \tag{7b}$$

where the inner-most summation extends over every branch of every support graph that meets the a criterion of the specified form. Algorithmically, this is implemented using a running tally for each $\eta(A \to BC)$ by examining each branch of each support graph in order to identify the balanced frequency to which it contributes.

4. *Re-estimate the probabilities:* Production rule probabilities are re-estimated using Eq. (5a).

The results produced by the gEM are the same as the results given by the IO algorithm [12]. While the IO passes through all the possible combinations of a grammar to produce a sequence, the graphical EM only uses the combinations given by the support graphs. Consequently, gEM is more efficient than IO in most practical cases, although the worst case time complexity per iteration is equal to that of IO. A greater memory complexity of $O(M_{nt}{}^3 L^2)$ is however needed to store the support graphs.

For reference, the algorithms for gEM are presented in Section 4.2 and Appendix A. Each iteration of the algorithms follow a structure that is analogous to the standard IO technique. The inside and explanation probabilities are computed using `Get-Inside-Probs()` (Algorithm 2). The outside probabilities and balanced

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

frequencies are computed using `Get-Expectations()` (Algorithm 3), while the routine shown in Algorithm 1 allows to re-estimates the production rule probabilities.

4. Derivations of graphical EM for on-line learning

In practical radar ES applications, the collection and analysis of representative data for learning of SCFG probabilities is expensive and time consuming. Only a limited amount of training data may be available to train a SCFG. It is however common to acquire new information from an operational environment or from other sources at different points in time, after the ES system has originally been deployed for operations. Moreover, in modern ES environments, the characteristics of MFR signals may vary dynamically in time. For timely and accurate recognition of threats, SCFG probabilities should be updated as new data becomes available.

In literature, the terms *incremental* and *on-line* are often used in reference to the type of learning described above. In the present paper, the term *on-line* is retained,[1] and an algorithm for supervised on-line learning possesses the following properties [18]. Training data should not be accumulated and stored in memory, and learning should not be performed from the start on all accumulated data. In addition, accommodating new training data should not corrupt previously acquired knowledge structure (*i.e.*, SCFG probabilities). It should not suffer from catastrophic forgetting, and thereby compromise ES system performance.

Since the performance a SCFG depends heavily on the availability of representative training data, on-line learning would provide the means to efficiently maintain an accurate and up-to-date model of an MFR of interest. It would also has the advantage of decreasing the memory requirements, since there is no need for storing the data from previous training phases. Furthermore, since training is performed only on the new training sequences, and not on all accumulated data, on-line learning would lower computational complexity. Finally, on-line learning may provide a powerful tool in a human-centric approach to radar ES, where domain experts are called upon to propose new data to gradually design and update MFR models as the operational environment unfolds.

Assuming that new radar signals are collected, analyzed and incorporated into blocks of training data, the following approach may be employed to design and maintain a SCFG that models the dynamics of a MFR. During the initial phase of SCFG design, the structure of the SCFG is described for the MFR system, and its production rule probabilities are initialized, either randomly, or based on prior domain knowledge. Then, the SCFG probabilities are learned based on an initial set of representative training sequences. As new blocks of training sequences are progressively gathered and analyzed for the theater of operation, or provided from other sources of electronic intelligence, on-line learning is performed to update and refine the SCFG probabilities of the MFR.

The classical EM-based techniques (IO and VS) are not suitable for on-line learning of SCFG probabilities. If new data becomes available, the SCFG must be re-trained using all cumulative data. Of the fast alternatives to IO and VS, only the HOLA [14] technique is suitable for learning new information in an on-line fashion – once it has trained the SCFG using an initial training set Ω, it can efficiently refine the summary statistics and SCFG probabilities to account for additional training set Ω'. The training for an augmented training set $\Omega \cup \Omega'$ can be completed by analyzing the new sequences in Ω' without having to re-compute the contributions from Ω. A brief summary of this approach applied to the learning of SCFG probabilities is presented in [6]. The other fast learning techniques, including gEM, are not on-line. In those cases, the additional training set Ω' can be incrementally incorporated into the data structures during pre-processing, but the iterative process must learn new probabilities from the beginning, using $\Omega \cup \Omega'$. The ability of HOLA, gEM or other techniques to preserve their previously acquired knowledge structure has not been addressed in literature.

The results from a comparison revealed that gEM can provide a significantly higher level of accuracy than HOLA [6] on radar data. Therefore, gEM may have great potential for preserving a SCFG's previously acquired knowledge structure during on-line learning. The rest of this section surveys EM algorithms for incremental and on-line learning, and describes two new versions of gEM that allow for efficient on-line learning of SCFG probabilities.

4.1. On-line and incremental learning with EM

Suppose that the values x of some random variable X has been observed, but not the values z of some other variable Z. The EM algorithm performs maximum likelihood estimation of parameters of an underlying distribution for problems in which some variables are not observed [19,20]. Assume the joint probability of X and Z is parameterized by θ, thus $P(x, z|\theta)$. Given observed data x, EM estimates the value of θ such that the log-likelihood of the marginal probability for X, $LL(\theta) = \log\{P(x|\theta)\} = \log\{\sum_z P(x, z|\theta)\}$, is maximized.

Let $\theta^{(m)}$ denote the value of parameter θ after m iterations. The EM algorithm starts with some initial guess of the model parameters $\theta^{(0)}$. Then it iteratively generates successive estimates $\theta^{(1)}, \theta^{(2)}, \ldots$ by applying the Expectation (E) and Maximization (M) steps repeatedly, until a stopping criterion is achieved. The mth iteration of the algorithm is as follows [21]:

1. **E-step:** given the known values x for the observed variables X and the current estimate of parameters $\theta^{(m)}$, compute the distribution $\tilde{P}^{(m)}$ over the range of unobserved variables Z, where $\tilde{P}^{(m)}(z) = P(z|x, \theta^{(m)})$. This allows to construct the conditional expectation for the complete data log-likelihood:
$$Q(\theta, \theta^{(m)}) = E_{\tilde{P}^{(m)}}[\log P(x, z|\theta)|x, \theta^{(m)}]. \quad (8)$$
2. **M-step:** re-estimate the parameters such that $\theta^{(m+1)} = \text{argmax}_\theta Q(\theta, \theta^{(m)})$.
3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \leq \varepsilon$, end iterations.

Each one of the iteration described above is guaranteed to increase the log-likelihood or leaves it unchanged. Wu [22] showed that this only ensured that EM would converge towards a stationary points. However, whether this stationary point is a local maximum, a global maximum, or an inflection point, depends on the initial parameters, and this issue is therefore easily handled by executing the algorithm several times, for a representative set of parameter combinations.

Note also that another stopping criterion is to use a condition based on the estimated parameters, such as the following [23]:
$$\frac{|\theta^{(m+1)} - \theta^{(m)}|}{|\theta^{(m)}|+\delta_1} < \delta_2 \quad (9)$$
As explained by Wu [22], convergence towards a combination of parameters implies convergence towards a likelihood value (under an assumption of continuity), while the opposite is not true. However, this kind of stopping criterion does not consider the issues of over-training, that are critical in on-line problems, since training is performed on limited data, while it can be handled by measuring a likelihood based measure on some validation data, as described in Section 5. That is why hereafter, to maintain coherence through the paper, a likelihood based criterion was considered.

[1] The term *incremental* alone will refer to a batch learning process that updates parameters using a subset of a training dataset, yet requires all the dataset to be stored in memory.

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

The batch learning EM algorithm described above requires the entire training dataset to be available at each iteration for updating the parameters, which can become an issue when the training dataset becomes large, or is partially available at a given moment. To bypass this limitation, several on-line, stochastic, incremental and sequential versions of EM have been proposed to update the hidden parameters when a single datum, or when a subset of the training dataset is available.

Neal and Hinton [21] proposed the *incremental EM* algorithm to accelerate the convergence of the EM algorithm. Assuming that a finite training data set Ω is divided into n blocks, each iteration of this algorithm performs a partial E-step (for a selected block) before performing the M-step on all the training data. 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. A variant of incremental EM that is relevant for on-line learning exploits the fact that parameters may be re-estimated using a vector of sufficient statistics for each block [21]. It consists in updating the sufficient statistics corresponding to a selected block of data during the E-step, and then re-estimating the parameters during the M-step, by combining the updated sufficient statistics with the existing ones (corresponding to the unused blocks). A drawback is that the M-step of this algorithm either requires all the training data or additional storage variables for all the training data. Although incremental EM is not suitable for on-line learning, the general approach has formed the basis for several efficient on-line learning algorithms, mainly for updating HMM parameters [24–27].

The recursive learning algorithm proposed by Titterington [28] is related to on-line estimation of parameters using EM. In particular, stochastic approximation procedures have been considered for the recursive estimation of parameters that can be linked to the EM algorithm [26]. This approach is adapted to EM as follows. A recursive approximation of Eq. (8) is computed during the E-step, and recursion is used to approximate the re-estimation of θ during the M-step. At iteration $m+1$, the recursion computes θ_{m+1} using the previous value $\theta^{(m)}$, a Fisher matrix corresponding to a complete observation of the data, and a vector of scores. Chung and Bohme [29] attempted to accelerate the convergence rate of this technique by proposing an adaptive procedure to determine the step size at each recursion. Based on Titterington's algorithm, Jorgensen [20] investigated a dynamic form of EM to re-estimate mixture proportions that requires a single EM update for each observation.

Sato and Ishii [30] proposed an on-line version of EM for normalized Gaussian networks that features a discount factor. The authors show that their on-line EM may be considered as a stochastic approximation method to find the ML estimator. It is based on an approximation from its previous value of the weighted means of parameters with respect to the posterior probability. These weighted means are then combined during the re-estimation phase to provide new estimations of the parameters.

On-line versions of the EM algorithm have recently been proposed for Hidden Markov Models (HMM), which can be seen as a very simple grammar. Andrieu and Doucet [31] use sufficient statistics to formulate an on-line Em algorithm for parameter estimation in general state-space models. Extending this idea, Mongillo and Deneve [32] developed an on-line version of EM for HMM, that allows to integrated new data without keeping memory of previously learned data, in the case where both the states and observations take a finite number of values. Cappé and Moulines [33] propose a recursive EM algorithm for latent variable models of independent observations. Vasquez et al. use a on-line algorithms for incrementally learning both the parameters and the structure of Growing Hidden Markov Models, in order to learn motion patterns. These two contributions led Cappé [34] to designing an on-line algorithm for general HMM, with possibly continuous observations. For more details on on-line learning techniques of HMM, Kreich et al. [35] made an extensive review of on-line techniques for estimating HMM parameters, comparing the different techniques in terms of performance and convergence.

4.2. On-line gEM algorithm (ogEM)

This section describes a new version of gEM for efficient on-line learning of SCFG probabilities that is called the *on-line gEM* (ogEM), and it is inspired by the incremental EM algorithm [21], and based on sufficient statistics[2] instead of raw data. Note that in the present context, the observed variables x_i corresponds to a given sequence, while the unobserved ones z_i corresponds to the associated derivation trees d_{x_i}.

4.2.1. Incremental EM using sufficient statistics

Assume that the original dataset $\Omega = \{x, z\}$ has been divided into n blocks $\{\Omega_1, \ldots, \Omega_n\} = \{x_1, z_1, \ldots, x_n, z_n\}$, where $\{x_1, \ldots, x_n\}$ is the set of observed values of variable X, and $\{z_1, \ldots, z_n\}$ is the set of unobserved values of variable Z. Then, a vector of sufficient statistics corresponding to each block Ω_j is set to an initial guess s_j, for $j = 1, 2, \ldots, n$. For the mth iteration, successive E and M steps are applied as follows:

1. **E-step:** select a block Ω_i to be updated.

$$\text{set } \tilde{s}_j^{(m+1)} = \tilde{s}_j^{(m)} \text{ for every } j \neq i. \tag{10}$$

$$\text{compute } \tilde{s}_i^{(m+1)} = E_{\tilde{P}_i}[s(x,z)] \quad \text{where } \tilde{P}_i(z) = P(z_i|x_j, \theta^{(m)}). \tag{11}$$

$$\text{set } \tilde{s}^{(m+1)} = \tilde{s}_i^{(m+1)} + \tilde{s}_j. \tag{12}$$

2. **M-step:** set $\theta^{(m+1)}$ to the maximum likelihood value of θ based on $\tilde{s}^{(m+1)}$.
3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \leq \varepsilon$, end iterations and store $\tilde{s}_i = \tilde{s}_i^{(m+1)}$

4.2.2. Incremental EM for SCFGs

Given the previous definition of sufficient statistics, η is defined as the vector of sufficient statistics of θ in gEM, and $\tilde{s}$ can be identified with η. The E-step now consists in computing the vector η, while the M-step consists in re-estimating the associated vector of production probabilities θ. As shown below, gEM can be modified to re-estimate SCFG production probabilities incrementally. For the mth iteration, after having divided the original dataset Ω into n blocks $\{\Omega_1, \ldots, \Omega_n\}$, each one containing a certain number of sequences, incremental EM may be adapted to gEM as follows:

1. **E-step:** select a block Ω_i to be updated.

$$\text{set } \eta_j^{(m+1)} = \eta_j^{(m)} \text{ for every } j \neq i. \tag{13}$$

$$\begin{aligned} &\text{compute } \eta_i^{(m+1)} \text{ using Algorithm 2 and 3 on } \Omega_i. \\ &\text{set } \eta^{(m+1)} = \eta_i^{(m+1)} + \eta_j^{(m+1)}. \end{aligned} \tag{14}$$

2. **M-step:** re-estimate $\theta^{(m+1)}$ using Eq. (5a) on $\eta^{(m+1)}$.
3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \leq \varepsilon$, end iterations and store $\eta_i = \eta_i^{(m+1)}$.

[2] Given a sample $\{x_1, \ldots, x_n\}$ 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. Accordingly, any inference strategy based on $f_\theta(x)$ may be replaced by a strategy based on $f_\theta(s)$ [36].

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

4.2.3. On-line gEM for SCFGs

With the incremental EM algorithm, all the data Ω is required to compute $\eta^{(m+1)}$. For on-line learning of a new block of data Ω_{n+1}, the following on-line approximation to the 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. To learn a new block Ω_2, the mth iteration of the E and M-steps of the on-line gEM (ogEM) algorithm is:

1. **E-step:**

$$\begin{array}{l}\text{compute } \eta^{(m+1)} \text{ using Algorithms 2 and 3 on } \Omega_2.\\ \text{set } \eta'^{(m+1)} = \eta^{(m+1)} + \eta_1.\end{array} \quad (15)$$

2. **M-step:** re-estimate θ using Eq. (5a).
3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \leq \varepsilon$, end iterations and store the new $\eta_2 = \eta'^{(m+1)}$.

The main difference with the incremental gEM algorithm lays in the fact that Ω_2 was not employed during the initial estimation of θ. If another dataset Ω_3 is available after training has been performed on $\{\Omega_0, \Omega_1, \Omega_2\}$, the procedure remains the same, using η_2 and applying the E-step on Ω_3.

A version of the ogEM algorithm is provided in Algorithm 1. Steps of this algorithm allow to manage updates to the value η. Note that in contrast to the original gEM, `Get-Inside-Probs()` is performed only on new data Ω_{i+1}, and `Get-Expectation()` produces $\eta^{(m+1)}$. Steps 2, 3, 7, 9, 10 and 14 highlight the difference between the algorithms of gEM, ogEM, and ogEM-df, described hereafter.

Algorithm 1 (`gEM()`, `gEM()`, `ogEM()`, `ogEM-df()`).

1- Initialize all parameters $\theta(A \to BC)$ such that $P(w^{(l)}|\theta) > 0$ for $l = 1, 2, ..., T$;
2- do nothing;
2- load η_i;
2- load $\tilde{N}_i$;
3- `Get-Inside-Probs()`;
3- `Get-Inside-Probs()` on Ω_{i+1};
3- `Get-Inside-Probs()` on Ω_{i+1};
4- $LL(\theta^{(0)} = log(\sum_{x\in\Omega} \alpha_x(0, L(x)|Start))$ using an independent validation dataset;
5- **while** $LL(\theta^{(m)}) - LL(\theta^{(m-1)}) > \varepsilon$ **do**
  6- `Get-Expectation()` (returns $\eta^{(m+1)}$);
  7- do nothing;
  7- $\eta'^{(m+1)} = \eta_i + \eta^{(m+1)}$;
  7- do nothing;
  8- **foreach** $(A \to \lambda) \in R$ **do**
    9- do nothing;
    9- do nothing;
    9- $\tilde{N}^{(m+1)}(A \to \lambda) = \tilde{N}_i(A \to \lambda) + \chi(m+1)[\frac{\eta^{(m+1)}(A\to\lambda)}{|\Omega_{i+1}|} - \tilde{N}_i(A \to \lambda)]$;
    10-$\theta'(A \to \lambda) = \eta(A \to \lambda)/\sum_{\lambda'} \eta(A \to \lambda')$;
    10- $\theta^{(m)}(A \to \lambda) = \eta'^{(m+1)}(A \to \lambda)/(\sum_{\lambda'} \eta'^{(m+1)}(A \to \lambda'))$;
    10- $\theta(A \to \lambda) = \tilde{N}^{(m+1)}(A \to \lambda)/(\tilde{N}^{(m+1)}(A))$;
  11- $m = m + 1$;
  12- `Get-Inside-Probs()`;
  13- $LL(\theta^{(m)}) = \sum_{x\in\Omega} \alpha_x(0, L(x)|Start)$ using an independent validation dataset;
14- do nothing;
14- store $\eta_{i+1} = \eta^{(m)}$;
14- store $\tilde{N}_{i+1}$;

4.3. On-line gEM with discount factor (ogEM-df)

In this section, the on-line EM algorithm of Sato and Ishii [30] is adapted for on-line learning of SCFGs using gEM. The new algorithm is called the *on-line gEM with discount factor* (ogEM-df), and can be seen as an extension of ogEM that allows for tuning the learning rate during on-line learning.

4.3.1. Batch EM for Gaussian networks

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

1. **E-step:** compute $P(z_i|x_i, \theta^{(m)})$, where $\theta^{(m)}$ is the estimation of parameter θ to estimate at iteration m;
2. **M-step:** compute the weighted mean over n samples of a function of parameters x with respect to the posterior probability:

$$f^*(x, z)^{(m)} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)P(z_i|x_i, \theta^{(m)}) \quad (16)$$

and re-estimate the parameters $\theta^{(m+1)}$ using $f^*(x, z)^{(m)}$. The reader is referred to [30] for more details on the re-estimation steps for this algorithm.
3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \leq \varepsilon$, end iterations.

4.3.2. Incremental EM for Gaussian networks

Based on the preceding algorithm, an incremental version can be derived. The principle of this incremental EM algorithm consists in estimating $f^*(x, z)$ iteratively. This estimate, $\tilde{f}(x, z)$, can be derived from $f(x, z)$ with:

$$\tilde{f}(x, z) = \chi\sum_{i=1}^{n}\left(\prod_{j=i+1}^{n}\xi(j)\right) f(x_i, z_i)P(z_i|x_i, \theta(i-1)) \quad \text{with}$$

$$\chi = \left(\sum_{i=1}^{n}\prod_{j=i+1}^{n}\xi(j)\right)^{-1} \quad (17)$$

where $\xi(j)$ is a time dependent discount factor, $0 \leq \xi(j) \leq 1$, 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}. The modified weighted mean of $f(x, z)$ can be computed iteratively by presenting the samples x_i sequentially, one after the other. Consider that $\tilde{f}(x_1^i, z_1^i)$ has already been computed on $\{x_1, \ldots, x_i, z_1, \ldots, z_i\}$:

$$\tilde{f}(x_1^{i+1}, z_1^{i+1}) = \tilde{f}(x_1^i, z_1^i) + \chi(i+1)(f(x_{i+1}, z_{i+1})P(z_{i+1}|x_{i+1}, \theta(i)) - \tilde{f}(x_1^i, z_1^i)) \quad (18)$$

It has been shown [30] that, with an appropriate value of χ, Eq. (18) is equal to Eq. (16), and that this incremental EM algorithm is equivalent to the batch version. Considering a dataset $\Omega = x_1, \ldots, x_n$, and the fact that training has already been completed on $\{x_1, \ldots, x_i\}$ the on-line EM algorithms can be applied as follows:

1. **E-step:** compute $P(z_{i+1} = j|x_{i+1}, \theta(i))$;
2. **M-step:** compute $\tilde{f}(x_1^{i+1}, z_1^{i+1})$ using Eq. (18) and re-estimate the parameters.
3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \leq \varepsilon$, end iterations.

4.3.3. Incremental gEM with discount factor for SCFGs

The incremental EM algorithm described above can be applied to learning of SCFGs by considering that x_i is a given sequence, and that z_i corresponds to d_{x_i}. 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

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

η for a particular production rule $A \to \lambda$ and a set of derivation trees $d_\Omega = \{d_x\}$:

$$\begin{aligned} N^*(A \to \lambda, d_\Omega) &= \frac{1}{|\Omega|}\sum_{x\in\Omega} N(A \to \lambda, d_x) P(d_x|x, G_s) \\ &= \frac{1}{|\Omega|}\sum_{x\in\Omega} N(A \to \lambda, d_x) \frac{P(x, d_x|G_s)}{P(x|G_s)} \\ &= \frac{1}{|\Omega|}\sum_{x\in\Omega} N(A \to \lambda, d_x) \frac{P(x, d_x|G_s)}{P(x, \Delta_x|G_s)} \\ &= \frac{\eta(A \to \lambda, d_\Omega)}{|\Omega|} \end{aligned} \tag{19}$$

All the derivation trees must however be considered to re-estimate the production rule probabilities. For each sequence, one must sum the weighted frequency of the rules over all the corresponding derivation trees as follows:

$$\begin{aligned} N^*(A \to \lambda) &= \sum_{d_\Omega \in \Delta_\Omega} N^*(A \to \lambda, d_\Omega) \\ &= \sum_{d_\Omega \in \Delta_\Omega} \frac{1}{|\Omega|}\sum_{x\in\Omega} N(A \to \lambda, d_x) \frac{P(x, d_x|G_s)}{P(x, \Delta_x|G_s)} \\ &= \frac{1}{|\Omega|}\sum_{x\in\Omega} \frac{1}{P(x, \Delta_x|G_s)} \sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x|G_s) \\ &= \frac{\eta(A \to \lambda)}{|\Omega|} \end{aligned} \tag{20}$$

Thus, the re-estimation equation (Eq. (5a)) can be modified according to Eq. (20):

$$\theta(A \to \lambda) = \frac{\eta(A \to \lambda)}{\eta(A)} = \frac{\eta(A \to \lambda)/|\Omega|}{\eta(A)/|\Omega|} = \frac{N^*(A \to \lambda)}{\sum_\lambda N^*(A \to \lambda)} \tag{21}$$

Assuming that $\tilde{N}_i(A \to \lambda)$ has already been computed from $\{x_1, \ldots, x_i\}$. It is now possible to compute $\tilde{N}_{i+1}(A \to \lambda)$ iteratively using Eq. (20), and the updated version of $N^*(A \to \lambda)$ is computed on $\{x_1, \ldots, x_i\}$, as follows:

$$\begin{aligned} \tilde{N}_{i+1}(A \to \lambda) &= \tilde{N}_i(A \to \lambda) + \chi(i+1)[\sum_{d_{x_{i+1}} \in \Delta_{x_{i+1}}} N(A \to \lambda, d_{x_{i+1}}) \\ &\quad \times P(d_{x_{i+1}}|x_{i+1}, G_s) - \tilde{N}_i(A \to \lambda)] \\ &= \tilde{N}_i(A \to \lambda) + \chi(i+1)[\eta(A \to \lambda, x_{i+1}) \\ &\quad - \tilde{N}_i(A \to \lambda)] \end{aligned} \tag{22}$$

where $\chi(i+1)^{-1}$ is a discount factor [26] and allows for tuning the learning rate of the algorithm. Setting $\chi(i+1)$ to 0 (*i.e.*, $\chi(i+1)^{-1} = 1$) means that no learning is performed, while setting $\chi(i+1)$ to 1 (*i.e.*, $\chi(i+1)^{-1} = 0$) means that previously learned blocks are no longer considered. Based on Eq. (22), this incremental EM algorithm can be adapted to gEM as follows. After training was completed on $\{x_1, \ldots, x_i\}$, the mth iteration gives:

1. **E-step:**

$$\begin{aligned} &\text{compute } \eta_{i+1} \text{ using Algorithms 2 and 3 on } x_{i+1}; \\ &\text{set } \tilde{N}_{i+1} = \tilde{N}_i + \chi(i+1)[\eta_{i+1} - \tilde{N}_i]; \end{aligned} \tag{23}$$

2. **M-step:**

$$\text{re-estimate } \theta^{(i+1)}(A \to \lambda) = \frac{\tilde{N}_{i+1}(A \to \lambda)}{\tilde{N}_{i+1}(A)}; \tag{24}$$

3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \le \varepsilon$, end iterations and store $\tilde{N}_{i+1}$.

4.3.4. On-line gEM with discount factor (ogEM-df)

Up till now, the algorithms described in this subsection are said to be incremental because they consists in presenting all the samples of a training dataset in sequence, from first to last, and then looping until convergence. However, it can be adapted to on-line learning of data as follows. Consider two data blocks $\{\Omega_1, \ldots, \Omega_i\}$ and Ω_{i+1} of sizes $|\Omega_{1,\ldots,i}|$ and $|\Omega_{i+1}|$ and suppose that after training on $\{\Omega_1, \ldots, \Omega_i\}$, $\tilde{N}_i(A \to \lambda)$ was already computed:

$$\begin{aligned} &\tilde{N}_{i+1}(A \to \lambda) \\ &= \tilde{N}_i(A \to \lambda) + \chi(i+1) \\ &\times \left[\frac{\sum_{x\in\Omega_{i+1}}\sum_{d_x\in\Delta_x} N(A \to \lambda, d_x) P(d_x|x, G_s)}{|\Omega_{i+1}|} - \tilde{N}_i(A \to \lambda)\right] \\ &= \tilde{N}_i(A \to \lambda) + \chi(i+1)\left[\frac{\sum_{x\in\Omega_{i+1}} \eta(A \to \lambda, x)}{|\Omega_{i+1}|} - \tilde{N}_i(A \to \lambda)\right] \\ &= \tilde{N}_i(A \to \lambda) + \chi(i+1)\left[\frac{\eta_{i+1}(A \to \lambda)}{|\Omega_{i+1}|} - \tilde{N}_i(A \to \lambda)\right] \end{aligned} \tag{25}$$

The on-line gEM with discount factor (ogEM-df) can be defined, based on ogEM. 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 $A \to \lambda$, the final value of $\tilde{N}_1(A \to \lambda)$ is stored. In order to learn a new block Ω_2, $\tilde{N}_2(A \to \lambda)$ can be computed for the mth iteration according to:

1. **E-step:**

$$\begin{aligned} &\text{compute } \eta^{(m+1)} \text{ using Algorithms 2 and 3 on } \Omega_2; \\ &\text{set } \tilde{N}^{(m+1)} = \tilde{N}_1 + \chi(m+1)\left[\frac{\eta_2^{(m+1)}}{|\Omega_2|} - \tilde{N}_1\right]; \end{aligned} \tag{26}$$

2. **M-step:**

$$\text{re-estimate } \theta'(A \to \lambda) = \frac{\tilde{N}^{(m+1)}(A \to \lambda)}{\tilde{N}^{(m+1)}(A)}; \tag{27}$$

3. **Stopping criterion:** if $LL(\theta^{(m+1)}) - LL(\theta^{(m)}) \le \varepsilon$, end iterations and store $\tilde{N}_2 = \tilde{N}^{(m+1)}$.

Algorithm 1 presents the ogEM-df technique. The steps of this algorithm allow to updates the value $\tilde{N}$. In contrast to the original gEM, `Get-Inside-Probs()` is performed only on new data Ω_{i+1}, and `Get-Expectation()` produces $\eta^{(m+1)}$. Steps 2, 3, 7, 9, 10 and 14 highlight the difference between the algorithms of gEM, ogEM, and ogEM-df.

4.4. Comparison of ogEM, ogEM-df and HOLA

The main difference between ogEM and ogEM-df lies in the fact that tuning parameter χ in ogEM-df adjusts the relative importance of the new dataset. It is straightforward to demonstrate that by setting $\chi(i+1) = \chi(i)/(1+\chi(i))$, these two algorithms become identical.

Note that using these two algorithms, consistency is ensured and local optima are likely to be avoided thanks to the random re-initialization of the SCFG probabilities when new data is added. Suppose that training was previously performed on a block Ω_1 and that a new one, Ω_2, becomes available. Both ogEM and ogEM-df 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 block is not fully representative of the whole phenomenon, and that their distributions are possibly disjoined in the input feature space. In this case, the production rules obtained after training on Ω_1 are

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

not the basis for training on Ω_2, and re-initialization is more likely to yield solutions that do not correspond to a local optimum.

As mentioned, HOLA [6,14] is a gradient-descent algorithm based on relative entropy that allows for on-line learning of SCFGs probabilities. When Ω_2 becomes available, an histogram representing the distribution of rules (created using Ω_1) is updated. When performing on-line learning of Ω_2, the SCFG probabilities obtained from training with Ω_1 constitute the starting point for the training process. HOLA seeks to reproduce the histogram using sequences produced by the grammar. Since the new histogram includes information on Ω_1, learning is not biased by using the production probabilities obtained from training with Ω_1. On the contrary, they represent a priori information, which allows HOLA to avoid local optima.

The time complexity per iteration and space complexity of ogEM and ogEM-df are identical to that of gEM since their complexity depends on the computations in `Get-Inside-Probs` and `Get-Expectation`. These are left unchanged during the on-line adaptation. Since HOLA uses a fixed size distribution, the time required for an iteration is independent of the size of the blocks, and it may therefore be significantly faster than ogEM and ogEM-df.

5. Experimental methodology

5.1. MFR radar data

Four fictitious but representative families of MFR systems, called Mercury, Pluto, VenusA and VenusB were considered in this paper. While a real application requires a complex analysis of received pulses in order to get a clean sequence of words, this aspect is out of the scope of this work, and it is considered that the sequences of words actually emitted by the radar are accessible.

The *Mercury MFR* has a vocabulary of 9 distinct words. It can be in one of five functional states, each state emitting a phrase of 4 words. The combinations of four-word phrases that are available to each state are given in [6]. The *Pluto MFR* has a vocabulary of 6 distinct words. It can be in one of four functional states, each state emitting a phrase of 3 words. The combinations of three-word phrases that are available to each state are given in [6].

Two word-level CFGs in Chomsky Normal Form were designed for these MFRs from their functional descriptions [37,6]. Datasets are generated for Mercury and Pluto, and consist of 400 sequences each, where each sequence corresponds to the set of words that would be produced by these MFRs during one target detection, while switching through all their internal states. Mercury sequences have a length that ranges from 108 to 1540 words, with an average of 588 words. Pluto sequences have a length that ranges from 397 to 953 words, with an average of 644 words.

The data sets of both MFRs were then partitioned into four equal parts – a training subset (`MTrain`/`PTrain`), a validation subset (`MVal`/`PVal`), a test subset (`MTest`/`PTest`), and an *ideal* test subset (`MTI`/`PTI`). 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 $\texttt{MTN}_2$ – and `PTI` – $\texttt{PTN}_1$ and $\texttt{PTN}_2$ – were generated. Each noisy test set consists of the 100 sequences from `MTI` and `PTI`. 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 $\texttt{PTN}_1$, and 10 words for $\texttt{MTN}_2$ and $\texttt{PTN}_2$. Finally, a different word of the vocabulary was randomly chosen to replace the each incorrectly detected word. For each sequence in each database, the corresponding states of the MFRs were stored for reference.

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, known as *VenusA* and *VenusB*, were simulated. The languages of VenusA and VenusB are described in details in [6]. Since these MFRs are only used for tests, and not for training, there was no need to design corresponding CFGs. In each case, a dataset of 100 sequences, named `VATI` for VenusA and `VBTI` for VenusB, were generated. During the simulations, the dwell durations of the different states were fixed, or set using a Poisson law, dependently of the state considered. At each state dwell, the choices of the corresponding emitted phrases was determined randomly over the possible phrases.

In radar ES applications, pulses are received from an illuminating radar and converted into 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 into several sequences in the languages of the MFRs of its library. A radar ES systems library may only contain 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 [6]. 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`).

A grammar is said to be *ambiguous* if several derivation trees can lead to the same sequence. In the radar ES application, MFRs grammars were constructed so that they are ambiguous if two consecutive states can produce a same phrase. The Pluto grammar is found to be 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 [6] that that their ambiguity would be null.

5.2. Experimental protocol for computer simulations

The protocol used in this paper involves on-line learning of SCFG probabilities and hold-out validation. Subsets `MTrain`, `MVal`, `PTrain`, and `PVal`, described in Section 5.1, are subdivided into 20, 10, 5, 4 or 2 subdatasets of 5, 10, 20, 25 or 50 sequences, resulting in new training and validation subdatasets. These sets will be referred herein as $\texttt{MTrain}_i(j)$, $\texttt{MVal}_i(j)$, $\texttt{PTrain}_i(j)$ and $\texttt{PVal}_i(j)$, where i indicates the size of the subdatasets ($i = 5$, 10, 20, 25 or 50), while j indicates the number of the subdataset. It is assumed that when a new block of training sequences Ω_j of i sequences becomes available, it is divided equally among a training $\texttt{MTrain}_i(j)$ ($\texttt{PTrain}_i(j)$) and a validation $\texttt{MVal}_i(j)$ ($\texttt{PVal}_i(j)$) subset.

Prior to each simulation trial, the SCFG probabilities are initialized randomly. During each trial with a given block size, batch learning with the ogEM and ogEM-df techniques is performed

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

on $\texttt{MTrain}_i(1)$ ($\texttt{PTrain}_i(1)$) using hold-out validation on $\texttt{MVal}_i(1)$ ($\texttt{PVal}_i(1)$), until the difference between the approximate negative log likelihoods of the sequences on $\texttt{MVal}_i(1)$ ($\texttt{PVal}_i(1)$) is lower than 0.001 for two consecutive iterations. At the end of each trial, the set of SCFG probabilities corresponding to the minima of the negative log likelihood of the sequences of $\texttt{MVal}_i$ ($\texttt{PVal}_i$) is kept. 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 $\texttt{MTrain}_i(1)$ ($\texttt{PTrain}_i(1)$) has been learned via batch learning, $\texttt{MTrain}_i(2)$ ($\texttt{PTrain}_i(2)$) is learned in an on-line fashion. With ogEM and ogEM-df, 10 new sets of SCFG probabilities are generated randomly. For on-line learning with HOLA, the 10 sets of SCFG probabilities from the previous iteration are used. Re-estimation of SCFG probabilities is performed for each set of probabilities based on the vectors of sufficient statistics stored after learning $\texttt{MTrain}_i(1)$ ($\texttt{PTrain}_i(1)$). The same process is repeated for successive blocks of training data, until $\sum_i|\texttt{MTrain}_i(1)| = 100$ ($\sum_i|\texttt{PTrain}_i(1)| = 100$). 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. 2 summarizes the experimental protocol used to assess performance.

5.3. Performance measures

The performance of learning techniques was compared in terms of the amount of resources required during training, and the accuracy of results on the test sets. The accuracy of the SCFGs produced using a learning technique was assessed in terms of the perplexity measure, the classification error rate over radar states, and ROC curves. The amount of resources required to implement these techniques is measured using the time and memory complexity and the convergence time. Estimation of the worst-case and average-case execution time per iteration and storage requirements allows for the assessment of computational complexity, while the remaining performance measures are assessed via computer simulation, using radar data sets. The outcome of numerous computer simulations are combined to yield an average performance measure. These measures are now briefly defined. (The reader is referred to [6] for detailed definitions.)

Perplexity (PP) is measured with $PP = 2^{-1/L \log_2 P(x)}$, where $P(x) = \sum_{d_x \in \overline{\Delta}_x} P(x, d_x|G_s)$ (Eq. (2)) with every possible parse tree) is the probability of the sequence x being produced by a language [38] and L is the length of x. Perplexity can be interpreted as the number of words that the SCFG has to choose from to complete a given sequence, and is based on information theory. The closer this value is to 1, the more a model can predict the language.

Classification error rate on estimated radar states is estimated by the ratio of incorrectly estimated states over all occurrences of the states in the test sets.

Given 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 produced by a SCFG and setting a decision threshold, a binary classification can be performed on each sequence. Four results can then be extracted – *True Positive* and *False Negative* for correctly and wrongly classified Positives, and *True Negative* and *False Positive* for correctly and wrongly classified Negatives. Varying the decision threshold allows to create an ROC curve [39] based on perplexity, which represents the *True Positive Rate* – or *Hit Rate* – versus the *False Positive Rate* – or *False Alarm Rate*. An indicator of performance is the area under this curve (AUC), which ranges from 0.5 (no discrimination) to 1.0 (full discrimination).

During a computer simulation, each complete presentation of the sequences in the training set is called an iteration. *Convergence time I* is measured by counting the number of iterations needed by a technique to learn Train before the termination condition is reached. Convergence of gEM and HOLA techniques is reached, respectively, once the perplexity or relative entropy remain approximately constant for two successive presentations of the training set.

Time complexity (T) can be estimated analytically from the time required during one iteration, to re-estimate production rule probabilities of a SCFG from a single training sequence. The result is a total average-case or worst-case running time formula, T, which summarizes the behavior of an algorithm as a function of key parameters, such as the number of sequences in the training dataset or their lengths. For simplicity, time complexity is measured by the total number of multiplications and divisions evaluated during each iteration. The *worst-case* represents the case in which all the possible productions are possible, whereas the *average-case* is estimated by introducing the stochastic parameters, some of which are specific to an algorithm. Finally, the *growth rate* is obtained by making the parameters of the worst-case complexity tend to infinity.

This last measure is independent from convergence time, since a technique 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 $T^* = T_{\text{init}} + I \cdot T \cdot \Gamma_T$, where T_{init} is the time required initially to produce data structures (such as support graphs or histograms), 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 training subset $|\texttt{MTrain}|(|\texttt{PTrain}|)$. It indicates the number of sequences a technique must process during the iterative process. For batch training with EM-based techniques (*i.e.*, IO, VS and gEM), $\Gamma_T = |\texttt{MTrain}|$ ($\Gamma_T = |\texttt{PTrain}|$), but it varies according to the technique with on-line training.

Memory complexity (M) 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 preprocessing phase was considered. For gEM, as described in Fig. 1, one branch of a support graph consists in 3 vectors of 3 registers (for example [*Start*, *Acq*, ϵ] ; [*Acq*, 0, 4] ; [ϵ, 4, 5], which means that the non-terminal *Start* is expanded by *Acq* ϵ to produce the subsequence $\{w_1, \ldots, w_5\}$), or 2 vectors of 2 and 3 registers ([*W*6, 6] ; [6, 0, 1], which means that *W*6 is expanded by to produce $w_1 = 6$), as shown in Fig. 1. For HOLA, one counting register is needed to represent the frequency $N(r).o$ for each rule $r \in R$.

The measure M contains limited information since only one sequence is considered. In order to measure the impact of on-line learning on memory requirements, as for the time complexity, the overall memory complexity M^* is $M^* = M \cdot \Gamma_M$ is computed, where M is the basic memory complexity, and Γ_M is a multiplicative factor that depends on the size of the training subset $|\texttt{MTrain}|(|\texttt{PTrain}|)$. It indicates the number of sequences a technique has to store during the iterative process. For batch training with EM-based techniques such as IO, VS and gEM, $\Gamma_M = |\texttt{MTrain}|$ ($\Gamma_M = |\texttt{PTrain}|$), but varies with the techniques for on-line training.

6. Results and discussions

Figs. 3 and 4 show the average perplexity achieved by a SCFG after on-line learning with ogEM, ogEM-df and HOLA of the Mercury and Pluto data, using different block sizes $|\texttt{MTrain}(i)|$ and

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

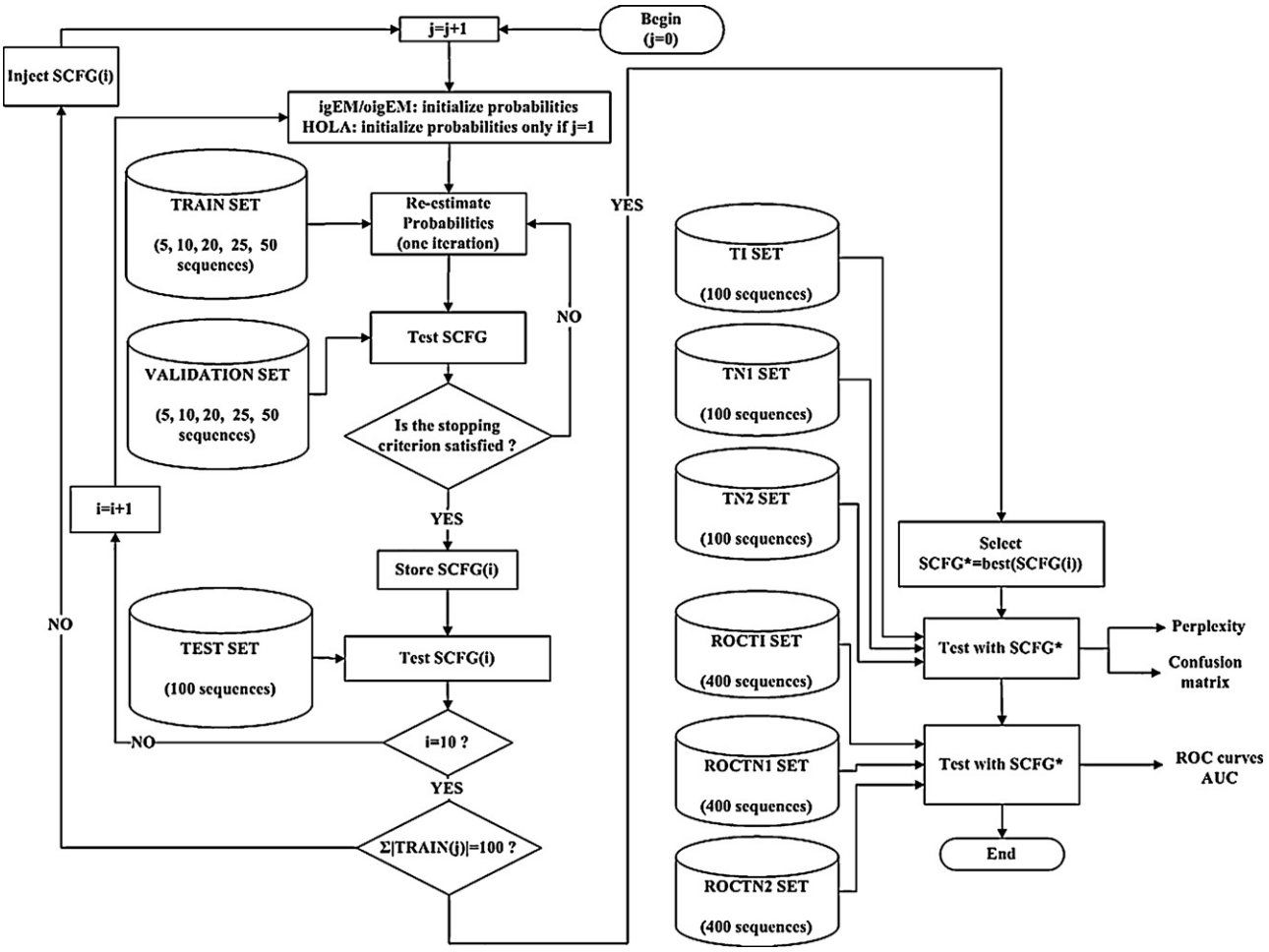


Fig. 2. Data flow diagram representation of the experimental protocol used to evaluate the performance of techniques for on-line learning of SCFG probabilities.

$|\mathtt{PTrain}(i)|$, respectively. With Mercury data, the perplexity of a SCFG obtained with either ogEM or ogEM-df tends towards the behavior that could be achieved through batch learning on one single cumulative block of data, even for small block sizes. It appears that $|\mathtt{MTrain}(i)| = 50$ sequences is required to reach the lowest possible perplexity. At that point, on-line learning with ogEM or ogEM-df gives the same performance as batch learning with gEM. In contrast, the perplexity of a SCFG obtained with HOLA never tends toward that of a SCFG trained though learning using one single cumulative block. It can be observed that the standard error of the sample mean remains very small for ogEM and ogEM-df, 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 with either ogEM or ogEM-df also tends toward the behavior of a SCFG obtained by batch learning with gEM. It appears that a $|\mathtt{PTrain}(i)| = 5$ sequences is sufficient to reach the lowest possible perplexity. At that point, on-line learning with ogEM or ogEM-df gives the same result as batch learning with gEM. HOLA requires a training block size of at least 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 ogEM-df has no significant impact on the overall perplexity when using training blocks larger than 10 sequences. For training block sizes of 5 and 10 sequences, the perplexity is more variable from trial to trial if χ is high, which is consistent with allocating greater importance to the new data in the learning process. As mentioned in Section 4.4, as blocks become smaller, each block contains fewer samples to represent the phenomenon, and their distributions are quite different in the input feature space. It can be observed that, for experiments with Pluto data, the standard error of the sample mean for the training on the first blocks of sequences for $|\mathtt{PTrain}(i)| = 5$ and 10 with ogEM and ogEM-df 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.

Fig. 5 displays an example of the average convergence time needed by ogEM-df($\chi = 0.25$) and HOLA to learn Pluto data, for different block sizes. The average convergence time on the Mercury data fluctuate in a range from 4 to 7 iterations across block sizes for ogEM and ogEM-df, whereas it ranges from 50 to 100 iterations for HOLA. This is consistent with the behavior of HOLA, since it has difficulty converging toward a minimum. A peak was observed for the second of the third block for ogEM-df($\chi = 0.25$), when $|\mathtt{MTrain}(i)| = 5$. Peaks do not appear for larger blocks, since they share a greater number of similar sequences from one block to another. From these results, one can conclude that the size of training blocks does not seem to have significant influence on the convergence time.

For experiments with Pluto data (shown in Fig. 5), the convergence time is higher than with Mercury data, and ranges from 4 to 16 iterations. As new blocks are added, the convergence time decreases for experiments with ogEM-df($\chi = 0.25$). However, the convergence time remains almost constant for ogEM-df($\chi = 0.50$),

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

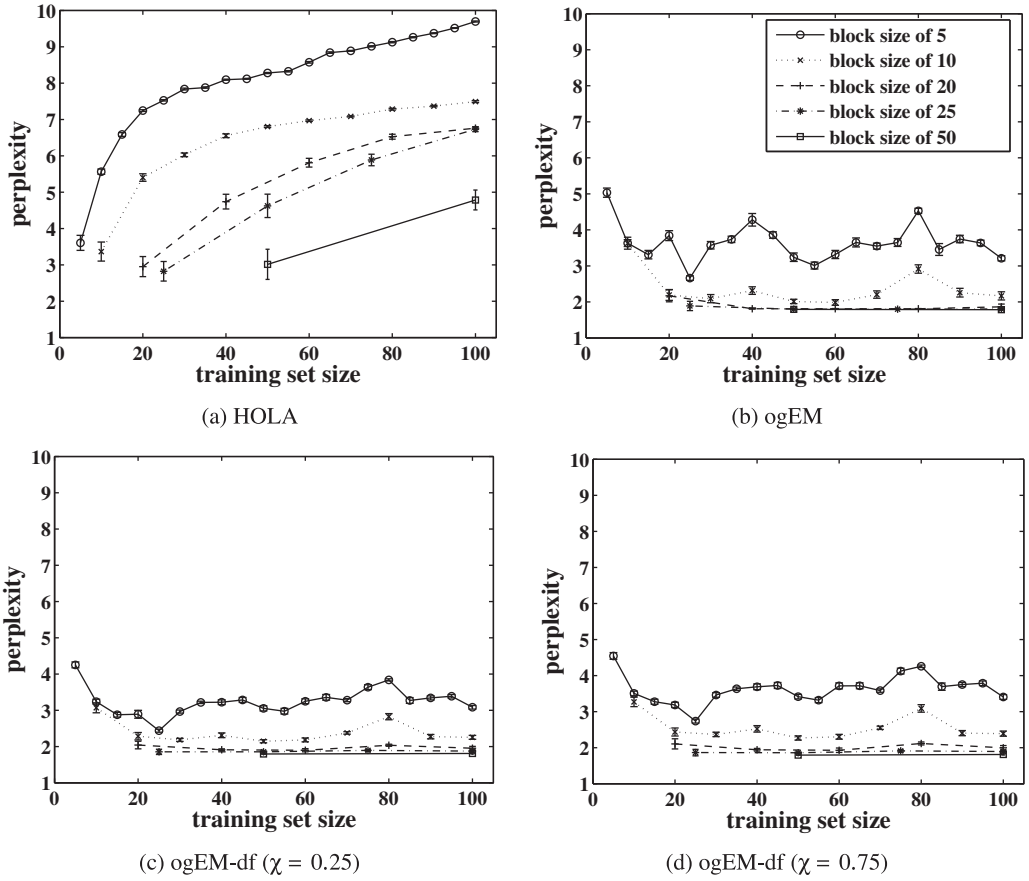


Fig. 3. Average perplexity of SCFGs obtained by on-line learning with HOLA, ogEM and iogEM on Mercury data for different block sizes. Error bars are standard error of the sample mean. Since ogEM and ogEM-df are equivalent to gEM when used for batch learning of a single block of data, the first point of each ogEM and ogEM-df curve also corresponds to the performance of gEM.

and increases for ogEM-df($\chi = 0.75$). The ogEM and ogEM-df techniques have similar convergence times when $\chi = 0.25$, although it tends to increase for ogEM-df when $\chi > 0.25$. As expected, giving more importance to new data implies that a greater number of iterations is needed to adapt to the new data. Experiments with HOLA show that by going from $|\mathtt{PTrain}(i)| = 10$ to $|\mathtt{PTrain}(i)| = 20$ significantly decreases the convergence time. This is consistent with the results displayed in Fig. 4(a), indicating that a minimum of 20 sequences per block is required in order to make HOLA converge.

Tables 1 and 2 show the confusion matrices of the SCFGs that obtain the lowest perplexity on MTI/PTI, $\mathtt{MTN}_1/\mathtt{PTN}_1$, and $\mathtt{MTN}_2/\mathtt{PTN}_2$ datasets, after training with $|\mathtt{MTrain}(i)|$=5 or $|\mathtt{PTrain}(i)| = 5$ sequences. It can be seen that SCFGs obtained after learning with ogEM and ogEM-df perform consistently well, while those obtained after learning with HOLA produce a considerable number of classification errors for TN1 and TN2. Moreover, HOLA is far more sensitive to situations in which two contiguous states can emit the same subsequence of words. This leads to the recognition rate per state shown for state S, and the confidence on estimated state shown for state Na.

For MTI, the estimation errors only occur in Table 1 when Na is estimated instead of Acq, while far more errors occur in Table 2. As discussed in Section 5.1, classification errors can only appear when several parse trees exist for a same sequence. Ambiguity emerges from the fact that Acq and Na can produce the same phrase. As with batch learning experiments, the ability of SCFGs to estimate radar states degrades with noisy data.

Table 3 shows the mean perplexity obtained for the SCFG that obtains the lowest perplexity on the MTI/PTI, $\mathtt{MTN}_1/\mathtt{PTN}_1$, and $\mathtt{MTN}_2/\mathtt{PTN}_2$ test sets. Although perplexity tends to grow with the level of noise for all three techniques, the perplexity obtained with ogEM and ogEM-df is always significantly lower than that of HOLA. All the algorithms give the same results for $\mathtt{PTN}_2$ (within the computer's limit on precision).

Fig. 6 shows the example of ROC curves for the Mercury grammar, using the SCFGs that obtain the lowest perplexity on MROCTI, $\mathtt{MROCTN}_1$, and $\mathtt{MROCTN}_2$ databases. Recall that in these databases, the first 100 sequences of MROCTI, $\mathtt{MROCTN}_1$, and $\mathtt{MROCTN}_2$ belong to Mercury language, whereas the last 300 sequences belong to the three other MFRs languages. It can be observed 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. The analysis of the perplexity obtained on the whole databases MROCTI, $\mathtt{MROCTN}_1$, and $\mathtt{MROCTN}_2$ shows that the errors come from sequences from VenusA and VenusB, since their language are quite similar to that of Mercury, while Pluto's language is more distinct from the three others. In results not presented here, ROC curves show that increasing noise to a frequency of one in 10 words for the replacements never has a significant impact on the performance obtained with Pluto SCFGs. Overall, it appears that the SCFGs can always detect the MFR of interest with very high level of accuracy. The worst results, given by HOLA on $\mathtt{MROCTN}_2$, correspond to an extreme case in which only the sequences emitted by the MFR of interest are noisy, with a high level of noise. Finally, Table 4 displays the decision threshold γ^* that corresponds to the best point of operation, when the difference between HR and FAR is maximized: $\gamma^* = \mathrm{argmax}\{HR(\gamma) - FAR(\gamma)\}$, and when the relative cost of errors is equal. For this point of

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

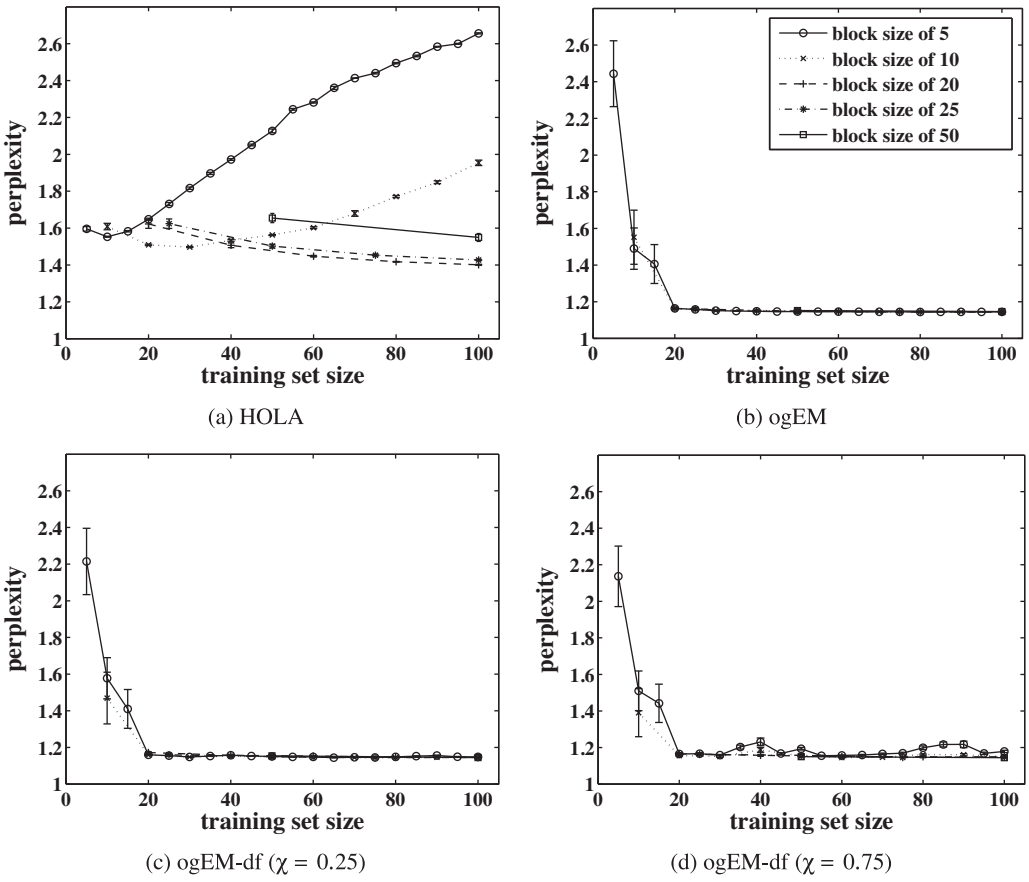


Fig. 4. Average perplexity of SCFGs obtained by on-line learning with HOLA, ogEM and iogEM on Pluto data for different block sizes. Error bars are standard error of the sample mean.

operation, HOLA requires a significantly higher FAR of 37% to produce a HR of 97%.

The importance for ogEM and ogEM-df of re-initializing the SCFG probabilities with each new block of data is shown in Figure 7, and results confirm the statements in Section 4.4. This figure shows the evolution of perplexity obtained with ogEM versus the training set sizes, after training on Mercury data when $|\texttt{MTrain}(i)| = 1$ sequence. Online 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 lower if the production rule probabilities are re-initialized. It indicates that ogEM is more likely to avoid local minima.

Tables 5 and 6 present the estimates of time complexity (T) per iteration and memory complexity (M) of the learning techniques used for on-line training. For one sequence, time complexity per iteration and memory complexity of gEM, ogEM, and ogEM-df only depend on the computational process of `Get-Inside-Probs` and of `Get-Expectation`, and these two procedures are left unchanged during the on-line learning of new data. Since the difference between gEM, ogEM and ogEM-df therefore lies only in the re-estimation equation, their time complexity per iteration and their memory complexity are the same.

In the presented experiments, the time (T) and memory (M) complexity required to train a SCFG may be computed in terms of

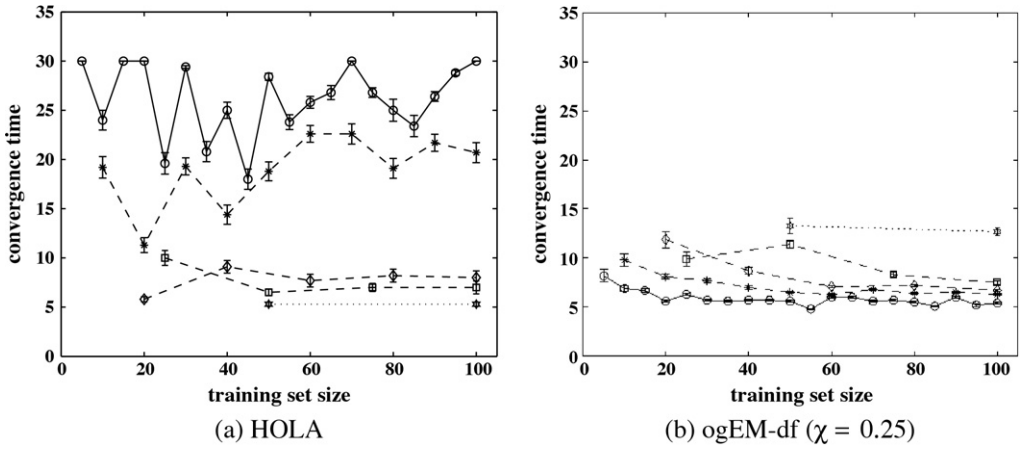


Fig. 5. Average convergence time SCFGs obtained by on-line learning with HOLA, and ogEM-df($\chi = 0.25$) on Pluto data for different block sizes. The results found in Fig. 4.

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

Table 1
Confusion matrix associated with $\mathtt{MTI}$, $\mathtt{MTN}_1$, and $\mathtt{MTN}_2$ for SCFGs obtained through on-line learning of $\mathtt{MTrain}$ with ogEM, ogEM-df and HOLA, using $|\mathtt{MTrain}(i)| = 5$. The classification rate per state are represented 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.

			Estimated states					Recognition rate per state (%)
			S $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Acq $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Na $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Rr $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Tm $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	
Real states		ogEM	4178/3174/2381	0/110/347	0/138/161	0/33/119	0/103/416	100/89.2/69.5
	S	ogEM-df	4178/3641/2273	0/121/360	0/114/222	0/18/142	0/101/422	100/91.1/66.5
		HOLA	4178/1043/840	0/473/522	0/159/188	0/133/202	0/153/398	100/53.2/39.1
		ogEM	0/10/32	1734/1511/948	235/333/350	0/9/60	0/22/76	88/80.2/64.7
	Acq	ogEM-df	0/8/45	1734/1516/926	235/332/364	0/10/54	0/13/79	88/80.7/63.1
		HOLA	0/0/23	1734/986/467	235/129/158	0/89/110	0/2/111	88/81.8/53.7
		ogEM	0/0/0	0/7/10	736/599/388	0/92/199	0/0/39	100/85.8/61.0
	Na	ogEM-df	0/0/0	0/8/27	736/608/383	0/82/193	0/0/40	100/87.1/59.36
		HOLA	0/0/2	0/10/7	736/399/335	0/158/99	0/0/21	100/70.4/72.2
		ogEM	0/31/34	0/0/32	0/79/167	2073/1802/1150	0/66/373	100/91.1/65.5
	Rr	ogEM-df	0/31/39	0/0/29	0/51/182	2073/1845/1133	0/56/358	100/93/65
		HOLA	0/2/49	0/3/47	0/373/422	2073/1129/592	0/168/354	100/67.4/40.4
		ogEM	0/759/1099	0/62/192	0/280/427	0/75/274	6243/5111/4272	100/81.3/68.2
	Tm	ogEM-df	0/485/1265	0/61/193	0/380/465	0/74/246	6243/5340/4094	100/84.2/65.4
		HOLA	0/2814/2342	0/149/363	0/690/852	0/200/580	6243/1915/1691	100/33.2/29
Confidence on estimated state (%)		ogEM	100/79.9/67.1	100/89.4/62	75.8/41.9/26	100/89.6/67.6	100/96.4/82.5	
		ogEM-df	100/87.4/62.8	100/88.8/60.3	75.8/40.9/23.7	100/90.9/53.9	100/96.9/82	
		HOLA	100/27/25.8	100/60.8/33.2	75.8/22.8/17.1	100/66.1/37.4	100/85.6/65.7	

Table 2
Confusion matrix associated with $\mathtt{PTI}$, $\mathtt{PTN}_1$, and $\mathtt{PTN}_2$ for SCFGs obtained through on-line learning of $\mathtt{PTrain}$ with ogEM, ogEM-df and HOLA, using $|\mathtt{PTrain}(i)| = 5$.

			Estimated states				Recognition rate per state (%)
			S $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Na $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Rr $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	Tm $\mathtt{TI}/\mathtt{TN}_1/\mathtt{TN}_2$	
Real states		ogEM	9041/12912/9856	5517/2728/1668	0/9/1042	1582/124/1671	56/81.9/69.2
	S	ogEM-df	9041/13034/9726	5517/2561/1663	0/9/1098	1582/151/1689	56/82.7/68.6
		HOLA	183/7/178	14375/14391/8642	0/579/2168	1582/435/2857	1.1/0.03/1.3
		ogEM	0/326/325	1845/1151/263	0/314/809	0/0/327	100/64.3/15.6
	Acq	ogEM-df	0/322/323	1845/1141/251	0/324/784	0/0/356	100/63.9/14.6
		HOLA	0/4/30	1845/1102/260	0/563/605	0/86/812	100/62.8/15.2
		ogEM	0/66/306	0/0/29	2016/1647/431	0/266/1061	100/83.2/23.6
	Na	ogEM-df	0/66/306	0/0/24	2016/1637/416	0/274/1073	100/82.8/22.9
		HOLA	0/25/54	0/104/392	2016/1242/421	0/591/932	100/63.3/23.4
		ogEM	2685/3245/4049	0/142/486	0/0/421	8928/8026/5535	76.9/70.3/52.8
	Tm	ogEM-df	2685/3250/4053	0/142/486	0/0/405	8928/8017/5520	76.9/70.3/54.3
		HOLA	17/7/197	2668/3896/4899	0/87/1400	8928/7171/3757	76.9/64.3/36.6
Confidence on estimated state (%)		ogEM	77.1/78/67.8	74.9/28.6/10.7	100/83.6/15.7	85/95.4/64.8	
		ogEM-df	77.1/78.2/67.5	74.9/29.7/10.4	100/83.1/15.4	85/95/64.9	
		HOLA	91.5/0.96/38.8	0.98/0.57/0.18	100/50.3/0.92	85/91.2/45	

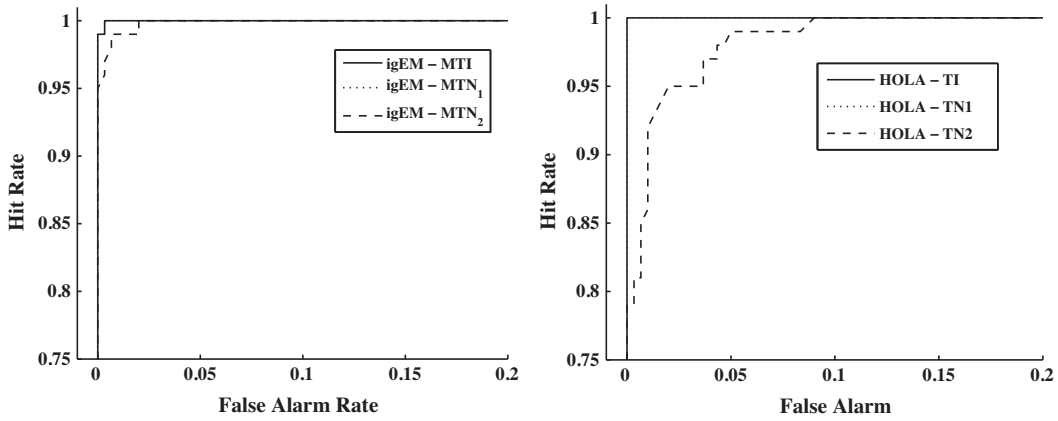
number of multiplication and divisions operations, and registers, respectively. The time and memory complexity associated with the SCFG model for Mercury are: $T_{HOLA} = 176 < T_{gEM/ogEM/ogEM-df} = 9.78 \times 10^3$ and $M_{HOLA} = 352 < M_{gEM/ogEM/ogEM-df} = 8.49 \times 10^3$, and the time and memory complexity associated with the SCFG model for Pluto are: $T_{HOLA} = 70 < T_{gEM/ogEM/ogEM-df} = 1.88 \times 10^4$ and $M_{HOLA} = 140 < M_{gEM/ogEM/ogEM-df} = 1.74 \times 10^4$. As expected, HOLA always results in the lowest time complexity per iteration and memory complexity.

The overall measures of time (T^*) and memory (M^*) complexity needed to learn a new block of training sequences would however reveal the true impact on performance of on-line learning. Consider that on-line learning algorithms are trained on a dataset $\mathtt{MTrain}$ ($\mathtt{PTrain}$), divided into n blocks $\mathtt{MTrain}(i = 1, \ldots, n)$ ($\mathtt{PTrain}(i = 1,$

Table 3
Perplexity of SCFGs obtained with ogEM, ogEM-df and HOLA on different test subsets.

Test subset	Mercury			Pluto		
	ogEM	ogEM-df	HOLA	ogEM	ogEM-df	HOLA
$\mathtt{MTI}/\mathtt{PTI}$	2.85	2.51	3.34	1.14	1.28	2.56
$\mathtt{MTN}_1/\mathtt{PTN}_1$	4.09	3.96	6.93	1.76	1.76	1.60
$\mathtt{MTN}_2/\mathtt{PTN}_2$	9.99	8.72	10.39	4.49	4.49	4.49

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022



(a) Mercury - ogEM: AUC = 0.999/0.999/0.999 (b) Mercury - HOLA: AUC = 1/1/0.996

Fig. 6. Examples of ROC curves for HOLA and ogEM from SCFGs that obtain the lowest perplexity on MROCTI, $\texttt{MROCTN}_1$ and $\texttt{MROCTN}_2$ data. Curves for ogEM and ogEM-df are almost identical.

Table 4
HR(γ^*) and FAR(γ^*) for SCFGs obtained with ogEM, ogEM-df and HOLA on MROCTI, $\texttt{MROCTN}_1$, and $\texttt{MROCTN}_2$.

	ogEM MROCTI / $\texttt{MROCTN}_1$ / $\texttt{MROCTN}_2$	ogEM-df MROCTI / $\texttt{MROCTN}_1$ / $\texttt{MROCTN}_2$	HOLA MROCTI / $\texttt{MROCTN}_1$ / $\texttt{MROCTN}_2$
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

Table 5
Estimates of time complexity per iteration of learning techniques. In this table, $|r|$ 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 subgraphs in support graph corresponding to transition and emission rules, $|\Delta l_t|$ is the average number of branches per subgraph corresponding to a transition rule. In order to compute the worst-case estimations, the following approximations have been considered: $|\varphi_t| = M_{nt} \cdot \sum_{i=1}^{L}\sum_{j=1}^{i-1} L = M_{nt} \cdot (L^2 - 1)$, $|\varphi_e| = M_{nt} \cdot L$, $|\Delta l_t| = M_{nt}^2 \cdot L$.

Techniques	Time complexity (T)		
	Average	Worst-case	Growth rate
gEM, ogEM and ogEM-df	$6\,|\varphi_e| + 9 \cdot |\varphi_t| \cdot |\Delta l_t|$	$6 \cdot M_{nt} \cdot L + 9 \cdot M_{nt}^3 \cdot L \cdot (L^2 - 1)$	$O(M_{nt}^3 \cdot L^3)$
HOLA	$2 \cdot |r|$	$2 \cdot (M_{nt}^3 + M_{nt} \cdot M_t)$	$O(M_{nt}^3)$

Table 6
Estimates of memory complexity of learning techniques. In this table, $|\Delta_l|$ is the average number of emission rules per tree. In order to compute the worst-case estimations, the following approximation has been considered: $|\Delta_l| = L$.

Techniques	Memory complexity (M)		
	Average	Worst-case	Growth rate
gEM, ogEM and ogEM-df	$L^2 \cdot M_{nt} \cdot (2 + |\Delta_l|) + 4 \cdot |\varphi_e| + 9 \cdot |\varphi_t| \cdot |\Delta l_t|$	$2 \cdot M_{nt} \cdot L \cdot (2 + L) + M_{nt}^3 \cdot (10 \cdot L^3 - 8)$	$O(M_{nt}^3 \cdot L^3)$
HOLA	$4 \cdot |r|$	$4 \cdot (M_{nt}^3 + M_t \cdot M_{nt})$	$O(M_{nt}^3)$

Table 7
Multiplicative factors for the computation of the overall time and space complexities associated with gEM, ogEM and ogEM-df techniques used for on-line learning. n is the total number of blocks. Train accounts for either MTrain or PTrain.

	gEM	ogEM/ogEM-df	HOLA
Γ_T	$\sum_{j=1}^{n}\sum_{i=1}^{j} |Train(i)|$	$\sum_{i=1}^{n} |Train(i)|$	Number of generated sequences
Γ_M	$\sum_{i=1}^{n} |Train(i)|$	$|\texttt{Train}(i)|$	1

. . ., n)), and that each new block is available once training was completed on the previous ones. The multiplicative factors for on-line learning techniques, Γ_T and Γ_M, are presented in Table 7. 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 ogEM and ogEM-df, 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 on-line learning provides a considerable saving in terms of computational resources. In addition HOLA provides the lowest overall time and memory complexities.

If the influence of preprocessing, T_{init}, is not considered, the overall time and memory complexity may be computed. Factor Γ_T takes the values 1050, 100, and 5, and Γ_T takes the values 100, 5, and 1, for gEM, ogEM/ogEM-df, and HOLA, respectively, when considering that on-line learning has been performed on a database of 100 sequences, with $|\texttt{MTrain}_i| = 5$ ($|\texttt{PTrain}_i| = 5$). Assume that HOLA also generates 5 sequences each time. Since M^* is independent from the convergence time, it has the same value for ogEM

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

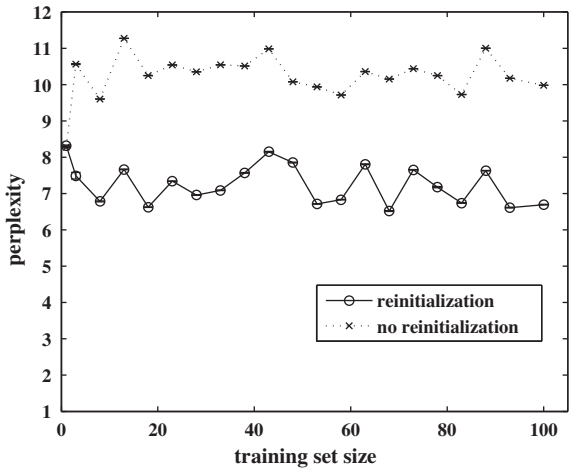


Fig. 7. Average perplexity obtained with ogEM versus the training sizes, after training on Mercury data with $|\texttt{MTrain}(i)| = 1$ sequence. On-line learning of blocks is performed with and without re-initializing the probabilities prior to learning each new block of training sequences.

and ogEM-df. Therefore, for the experiments on Mercury data, T^* and M^* are:

$$T^*_{HOLA} = 7.05 \times 10^4 < T^*_{ogEM} = 4.69 \times 10^6 < T^*_{ogEM-df(\chi=0.25)} = 4.79 \times 10^6, \ldots, < T^*_{ogEM-df(\chi=0.75)} = 4.89 \times 10^6 < T^*_{ogEM-df(\chi=0.50)} = 4.99 \times 10^6 < T^*_{gEM} = 5.13 \times 10^7;$$

$$M^*_{HOLA} = 14.08 \times 10^3 < M^*_{ogEM/ogEM-df} = 2.18 \times 10^8 < M^*_{gEM} = 4.36 \times 10^9.$$

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

$$T^*_{HOLA} = 3.15 \times 10^4 < T^*_{ogEM} = 9.40 \times 10^6 < T^*_{ogEM-df(\chi=0.25)} = 11.28 \times 10^6, \ldots, < T^*_{ogEM-df(\chi=0.50)} = 14.10 \times 10^6 < T^*_{ogEM-df(\chi=0.75)} = 18.80 \times 10^6 < T^*_{gEM} = 98.70 \times 10^6;$$

$$M^*_{HOLA} = 7.00 \times 10^3 < M^*_{ogEM/ogEM-df} = 1.12 \times 10^8 < M^*_{gEM} = 2.24 \times 10^9.$$

Results show that HOLA still has the lower overall time T^* and memory M^* complexities. However, ogEM and ogEM-df can be significantly faster to re-estimate the probabilities of a SCFG than gEM. 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.

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 their instantaneous level of threat. In modern environments, the 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, Stochastic Context Free Grammars (SCFGs) have been proposed for modeling the signals produced by 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. Unfortunately, application of the popular Inside–Outside (IO) and Viterbi Score (VS) techniques 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, efficient on-line learning of SCFG probabilities is important for rapidly reflecting changes in operational environments.

Two fast techniques for learning SCFG probabilities, named gEM and HOLA, were previously compared [6]. The gEM techniques is a variant of IO, that uses results from pre-processing to reduce the time complexity per iteration, whereas HOLA is an on-line technique based on gradient descent. In this paper, two new on-line derivations of gEM, called on-line gEM (ogEM) and on-line gEM with discount factor (ogEM-df), 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 in a on-line fashion, without accumulating and storing all training sequences, and without re-training a SCFG from the start using all cumulative data. An experimental protocol has been defined to assess impact on performance of factors such as the size of training blocks, and levels of grammar ambiguity. Proof-of-concept computer simulations have been performed on using synthetic radar 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 on-line learning of data blocks with ogEM and ogEM-df provides the same level of accuracy as learning all cumulative data from scratch, even for small data blocks and for complex grammars. As expected, on-line learning significantly reduces the overall time and memory complexities. The ogEM and ogEM-df algorithms systematically provide a higher level of accuracy than HOLA on the radar data, especially for small block of data, in terms of perplexity and estimation error over radar states. Assuming the MFR is not modeled by a very ambiguous SCFG, all techniques can efficiently learn SCFG probabilities at the expense of significantly higher memory resources. The execution time and memory requirements of HOLA are orders of magnitude lower than that of ogEM and ogEM-df.

Other relevant areas of application for ogEM and ogEM-df include bio-informatics, video content analysis, and character and speech recognition. Heeringa and Oates [40] compared HOLA to IO in a variety of application and observed that performance was similar. They used an English phrase grammar, a palindrome-generating grammar, and a simple ambiguous grammar. The main difference between these grammars and the ones used to model MFR lies in its underlying structure, and in particular the transition rules between non-terminal symbols. The English phrase grammar has the largest vocabulary (16 words), followed by the MFR grammars (5 words), and then the palindrome-generating and the simple ambiguous grammars (2 words). However, the Mercury grammar contains 40 non-terminal symbols and uses 88 different transition rules, the Pluto grammar contains 18 non-terminal symbols and uses 29 different transition rules, while the English phrase grammar contains only 6 non-terminal symbols and uses 17 different transition rules, the palindrome-generating grammar contains 3 non-terminal symbols and uses 6 different transition rules, and the

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022

simple ambiguous grammar contains 3 non-terminal symbols and uses 4 different transition rules. Therefore, a more exhaustive comparison of the three on-line algorithms compared in this paper, over a wider variety of grammars and applications, would probably reveal that HOLA is more suitable for simpler grammars and large samples, while ogEM and ogEM-df would be more suited for complex grammars. Indeed, all three algorithms would probably perform similarly, while HOLA's complexity is only bounded by the grammar complexity.

Appendix A. Description of the gEM algorithm.

Algorithm 2 (`Get-Inside-Probs()`).

1- **for** l=*1 to* $|\Omega|$ **do**
  2- Put $\delta_l = <\tau_1, ..., \tau_{|\delta_l|}>$;
  3- **for** $k = |\delta_l|$ *to 1* **do**
    4- **foreach** $E \in \varphi(\tau_k)$ **do**
      5- $\alpha^l(\tau_k, E) = 1$;
      6- **foreach** $e \in E$ **do**
        7- **if** $e = (A \rightarrow \lambda)$ **then**
          8- $\alpha^l_E(\tau_k) * = \theta(A \rightarrow \lambda)$;
        else
          9- $\alpha^l_E(\tau_k) * = \alpha^l(e)$;
    10- $\alpha^l(\tau_k) := \sum_{E \in \varphi(\tau_k)} \alpha_E(\tau_k, E)$;

Algorithm 3 (`Get-Expectations()`).

1- **foreach** $(A \rightarrow \lambda) \in \alpha$ **do**
  2- $\eta(A \rightarrow \lambda) = 0$;
3- **for** l=*1 to* $|\Omega|$ **do**
  4- Put $\delta_l = <\tau_1, ..., \tau_{\delta_l}>$;
  5- $\beta^l(\tau_1) := 1$;
  6- **for** k=*2 to* $|\delta_l$ **do**
    7- $\beta^l(\tau_k) := 0$;
  8- **for** k=*1 to* $|\delta_l|$ **do**
    9- **foreach** $E \in \varphi(\tau_k)$ **do**
      10- **foreach** $e \in E$ **do**
        11- **if** $e = (A \rightarrow \lambda)$ **then**
          12- $\eta(A \rightarrow \lambda) + = \beta^l(\tau_k) * \alpha^l_E(\tau_k) / \alpha^l(S(0, n_l))$;
        else
          13- **if** $\alpha^l(e) > 0$ **then**
            14- $\beta^l(e) + = \beta^l(\tau_k) * \alpha^l_E(\tau_k) / \alpha^l_E(e)$;

References

[1] K.S. Fu, Syntactic Pattern Recognition and Applications, Prentice-Hall, NJ, 1982.
[2] N. Visnevski, V. Krisnamurthy, A. Wang, S. Haykin, Syntactic modeling and signal processing of multifunction radars: a stochastic context-free grammar approach, Proceedings of the IEEE 95 (5) (2007) 1000–1025.
[3] A. Wang, V. Krishnamurthy, Signal interpretation of multifunction radars: modeling and statistical signal processing with stochastic context free grammar, IEEE Transactions on Signal Processing 56 (3) (2008) 1106–1119.
[4] K. Lari, S. Young, The estimation of Stochastic Context-Free Grammars using the Inside–Outside Algorithm, Computer Speech and Language 4 (1990) 35–56.
[5] H. Ney, Stochastic grammars and pattern recognition, speech recognition and understanding, Recent Advances, Trends, and Applications 75 (1992).
[6] G. Latombe, E. Granger, F.A. Dilkes, Fast learning of grammatical probabilities in radar Electronic Support, Tech. Rep. 2009-053, Defence R&D Canada – Ottawa, November 2009.
[7] J. Earley, An efficient context-free parsing algorithm, Communication of the ACM 13 (2) (1970) 84–102.
[8] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, 2nd ed., Addison-Wesley, 2001.
[9] A. Nijholt, The CYK-approach to serial and parallel parsing, Quarterly Journal of Linguistics of the Language Research Institute of Seoul National University 27 (2) (1991) 229–254.
[10] T. Fujisaki, F. Jelinek, J. Cocke, E. Black, T. Nishino, A probabilistic parsing method for sentence disambiguation, in: Int'l W. on Parsing Techn., Pittsburgh, USA, 1989, pp. 85–94.
[11] D.-Y. Ra, G.C. Stockman, A new one-pass algorithm for estimating Stochastic Context-Free Grammars, Information Processing Letters 72 (1999) 37–45.
[12] T. Sato, Y. Kameya, Parameter learning of logic programs for symbolic-statistical modeling, Journal of Artificial Intelligence Research 2 (2001) 391–454.
[13] A. Stolcke, An efficient probabilistic context-free parsing algorithm that computes prefix probabilities, Computational Linguistics 21 (2) (1995) 165–201.
[14] T. Oates, B. Heeringa, Estimating grammar parameters using bounded memory, in: H.F.P. Adriaans, M. van Zaanen (Eds.), Proceedings of the Sixth International Colloquium on Grammatical Inference, 2002, pp. 185–198.
[15] N. Visnevski, F.A. Dilkes, S. Haykin, V. Krisnamurthy, B. Currie, Non-self-embedding context-free grammars for multi-function radar modeling – electronic warfare application, in: Proc. of the 2005 IEEE Int. Radar Conf., Arlington, USA, 2005, pp. 669–674.
[16] R. Dowell, S. Eddy, Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction, BMC Bioinformatics 5 (1) (2004) 71.
[17] F. Nevado, J.A. Sanchez, J.M. Benedi, Combination of estimation algorithms and grammatical inference techniques to learn Stochastic Context-Free Grammars, in: Proc. of the 5th International Colloquium, ICGI 2000, Lisbon, Portugal, 2000, pp. 164–171.
[18] R. Polikar, L. Upda, S. Upda, V. Honavar, Learn++: an incremental learning algorithm for supervised neural networks, IEEE Transactions oN Systems Man and Cybernetics C 31 (4) (2001) 497–508.
[19] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum Likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society 33 (1) (1977) 1–38.
[20] M. Jorgensen, A dynamic EM algorithm for estimating mixture proportions, Statistics and Computing 9 (1999) 299–302.
[21] R. Neal, G. Hinton, A view of the EM algorithm that justifies incremental, sparse, and other variants, Learning in Graphical Models 89 (1998) 355–368.
[22] C. Wu, On the convergence properties of the em algorithm, The Annals of Statistics (1983) 95–103.
[23] J. Booth, J. Hobert, Maximizing generalized linear mixed model likelihoods with an automated monte carlo em algorithm, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 61 (1) (1999) 265–285.
[24] Y. Gotoh, M.M. Hochberg, H.F. Silverman, Efficient training algorithms for HMMs using incremental estimation, IEEE Transactions on Speech and Audio Processing 6 (6) (1998) 539–548.
[25] V.V. Digalakis, Online Adaptation of Hidden Markov Models using incremental estimation algorithms, IEEE Transactions on Speech and Audio Processing 7 (3) (1999) 253–261.
[26] S. Ng, G. McLachlan, A. Lee, An incremental EM-based learning approach for on-line prediction of hospital resource utilization, AI in Medicine 36 (3) (2006) 257–267.
[27] G. Mongillo, S. Deneve, Online learning with HMMs, Neural Computation 20 (7) (2008) 1706.
[28] D.M. Titterington, Recursive parameters estimation using incomplete data, Journal of Royal Statistic Society 46 (1984) 257–267.
[29] P.J. Chung, J.F. Bohme, Recursive EM algorithm with adaptive step size, in: 11th Int. Symp. on Signal Processing and its Applications, 2003, vol. 2, Paris, France, 2003, pp. 519–522.
[30] M. Sato, S. Ishii, On-line EM algorithm for the normalized Gaussian network, Neural Computation 12 (2) (2000) 407–432.
[31] C. Andrieu, A. Doucet, Online expectation-maximization type algorithms for parameter estimation in general state space models, in: 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings (ICASSP'03), IEEE, vol. 6, 2003, pp. VI-69.
[32] G. Mongillo, S. Deneve, Online learning with Hidden Markov Models, Neural computation 20 (7) (2008) 1706–1716.
[33] O. Cappé, E. Moulines, On-line expectation-maximization algorithm for latent data models, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 71 (3) (2009) 593–613.
[34] O. Cappé, Online EM algorithm for Hidden Markov Models, Available from: http://arxiv.org/abs/0908.2359, 2009.
[35] W. Kreich, E. Granger, A. Miri, R. Sabourin, A survey of techniques for incremental learning of HMM parameters, Information Sciences.
[36] L. Scharf, C. Demeure, Statistical Signal Processing: Detection, Estimation, and Time Series Analysis, Addison-Wesley Pub. Co, 1991.
[37] S. Haykin, B. Currie, Automatic radar emitter recognition using Hidden Markov Models, Unpublished Technical Report, DRDC Ottawa, 2003.
[38] Y. Estève, Intégration de sources de connaissances pour la modélisation stochastique du langage appliquée à la parole continue dans un contexte de dialogue oral homme-machine, Ph.D. Thesis, Université d'Avignon et des pays de Vaucluse, 2002.
[39] T. Fawcett, An introduction to ROC analysis, Pattern Recognition Letters 27 (8) (2006) 861–874.
[40] B. Heeringa, T. Oates, Two algorithms for learning parameters of Stochastic Context-Free Grammars, in: Working Notes of the 2001 AAAI Fall Symposium on Using Uncertainty within Computation, 2001.

Please cite this article in press as: G. Latombe, et al., Graphical EM for on-line learning of grammatical probabilities in radar Electronic Support, Appl. Soft Comput. J. (2012), doi:10.1016/j.asoc.2012.02.022