Computer Speech and Language (1996) **10,** 265–293

Stochastic automata for language modeling

Giuseppe Riccardi,† Roberto Pieraccini and Enrico Bocchieri

AT&T Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A. email: dsp3/roberto/enrico@research.att.com

Abstract

Stochastic language models are widely used in spoken language understanding to recognize and interpret the speech signal: the speech samples are decoded into word transcriptions by means of acoustic and syntactic models and then interpreted according to a semantic model. Both for speech recognition and understanding, search algorithms use stochastic models to extract the most likely uttered sentence and its correspondent interpretation. The design of the language models has to be effective in order to mostly constrain the search algorithms and has to be efficient to comply with the storage space limits.

In this work we present the Variable *N*-gram Stochastic Automaton (VNSA) language model that provides a unified formalism for building a wide class of language models. First, this approach allows for the use of accurate language models for large vocabulary speech recognition by using the standard search algorithm in the one-pass Viterbi decoder. Second, the unified formalism is an effective approach to incorporate different sources of information for computing the probability of word sequences. Third, the VNSAs are well suited for those applications where speech and language decoding cascades are implemented through weighted rational transductions. The VNSAs have been compared to standard bigram and trigram language models and their reduced set of parameters does not affect by any means the performances in terms of perplexity. The design of a stochastic language model through the VNSA is described and applied to word and phrase class-based language models. The effectiveness of VNSAs has been tested within the Air Travel Information System (ATIS) task to build the language model for the speech recognition and the language understanding system. © 1996 Academic Press Limited

1. Introduction

The *noisy channel* paradigm is the basis for the classical equation of speech recognition where a word sequence $W = w_1, w_2, \ldots, w_N$ is interpreted as a *noisy* version of the speech signal. Within this framework, given a sequence of acoustic measurements $O =$

† Author for correspondence.

0885–2308/96/040265 + 29 $25.00/0 © 1996 Academic Press Limited

$o_1, o_2, \ldots, o_M$, it has to be found a sequence W such that the *a posteriori* probability:

$$P(W|O) = \frac{P(O|W)P(W)}{P(O)} \tag{1.1}$$

is maximum over all possible sequences of words W:

$$\hat{W} = \arg \max_{W} P(O|W)P(W) \tag{1.2}$$

where $\hat{W}$ is the recognized sequence of words and $P(O|W)$ is the probability of a set of acoustic observations for a given sequence of words (Rabiner & Juang, 1993). The same framework has been applied successfully for machine translation (Brown *et al.*, 1990) and language understanding (Pieraccini & Levin, 1992) where, given a word sequence W, the most likely sequence of concepts $\hat{C}$ (interpretation) is found by maximizing the *a posteriori* probability $P(C|W)$ over all possible concepts C.

Stochastic language models are generally used for computing the probabilities $P(W)$ and $P(C)$ for any possible sequence of symbols (words or concepts). In order to perform the maximization of the type in Equation (1.2), one of the most successful parsing algorithms is the Viterbi decoding along with the beam search (Rabiner & Juang, 1993). Despite the advantage of being a time-synchronous search, only very simple word boundary constraints (e.g. word pair or bigram) have been exploited into such a decoding schema. A second approach is the *stack* decoding algorithm: it has the attractive property of being a *best first search* through a linguistic hypothesis tree and a search for the optimal word string rather than the optimal state sequence (as in the Viterbi algorithm) can be performed. However, the problems arising in calculating heuristic functions to prune all word string hypotheses make this algorithm difficult to use for speech recognition application (Bahl, Jelinek & Mercer, 1983). More recently, multi-step rescoring procedures have been designed to handle language models of increasing complexity (Soong, 1990; Austin, Schwartz & Placeway, 1991). In this case, first a one-pass Viterbi decoding is performed with a coarse language model (e.g. bigram) and a pruned lattice of word hypotheses is obtained: then, more accurate language models can be adopted to parse the *n-best* hypotheses. The problem with this approach is that several pruning stages can affect the search for the *best* word sequence while the best strategy would be to apply in a time-synchronous fashion the best available language model in a one-step decoding process.

The main motivation for our work is to be able to handle accurate language models in a one-step procedure of the maximization process in Equation (1.2) so that we can maximize the number of constraints and increase the accuracy. In order to achieve this goal we provide algorithms to design stochastic finite state automata that are both effective (in terms of performance) and efficient (in terms of parameter number) for all applications where a network search is performed (e.g. Viterbi decoder) or speech and language decoding cascades are implemented through weighted rational transductions (Pereira, Riley & Sproat, 1994). Furthermore, in this work we show how this unified formalism is an effective approach to incorporate different sources of information for computing the probability of word sequences (e.g. word and phrase class-based language models). Within the framework of the Variable N-gram Stochastic Automatons (VNSAs),

we design and evaluate class-based language models that outperform (in terms of perplexity and recognition error rate) the standard *n*-gram language model.

In the following section we describe the main issues related to the design of a stochastic language model and we recall the standard approach to *n*-gram language modeling. In Section 3 we describe the problem of representing a stochastic language model through a Stochastic Finite State Automaton. Then, in Section 4 we present our solution to approximate an *n*-gram language model through a finite state automaton: the VNSA. The computation of transition probabilities for these stochastic automata is detailed in Section 4.2. In Section 5 we study the impact of the VNSA's non-determinism on the language perplexity, and the word error rate (WER). Last, but not least, the design of a word and phrase class-based stochastic language model is presented in Section 6 and the results in terms of perplexity and WER for a one-pass Viterbi decoder are reported in Section 7, along with a comparison to the standard *n*-gram language models. The complete description of the application of these stochastic networks into a language understanding system is given in Pieraccini and Levin (1992, 1995).

2. The *n*-gram language and related issues

Stochastic language models are generally used for computing the probability $P(W)$ for any possible sequence of words $W=w_1, \ldots, w_N$. $P(W)$ can be written as:

$$P(W)=\prod_{i=1}^{N} P(w_i|w_1, \ldots, w_{i-1}) \tag{2.1}$$

In practice, it is impossible to estimate all the conditional probabilities $P(w_i|w_1, \ldots, w_{i-1})$ for a reasonable sentence length. In the literature, the most established model has been the *n*-gram language model (Jelinek & Mercer, 1980). In this case, it it assumed that the probability of a word w_i given the context $w_1, \ldots, w_{i-1}$, depends only on the n previous words. Thus, Equation (2.1) can be rewritten as:

$$\begin{aligned} P(W) &= \prod_{i=1}^{N} P(w_i|w_1, \ldots, w_{i-1}) \\ &= P(w_1, \ldots, w_{n-1}) \prod_{i=n}^{N} P(w_i|w_{i-n+1}, \ldots, w_{i-1}) \end{aligned} \tag{2.2}$$

The conditional probabilities of an *n*-gram model are generally estimated using a corpus of sentences that reflects the statistics of the language (training set). Under the Maximum Likelihood paradigm the estimated probabilities are of the form:

$$\hat{P}_{ML}(w_i|w_1, \ldots, w_{i-1})=\frac{C(w_1, \ldots, w_{i-1}, w_i)}{C(w_1, \ldots, w_{i-1})} \tag{2.3}$$

where $C(c_k)$ is the frequency of the event (word tuple) c_i in the corpus. In order to obtain a non-zero probability for any event $(w_1, \ldots, w_{i-1}, w_i)$, we have to take into

account rare or unseen events and estimate their probability on the basis of the available samples.[1] The most used approach to this problem has been the discounting of ML estimates. The probability mass of the observed events is discounted so that a probability mass can be assigned to the unseen events. Namely, the discounted probability $\hat{P}(c_k)$ is computed as a linear combination of the ML estimate:

$$\hat{P}(c_k) = a_k \hat{P}_{ML}(c_k) + b_k \quad a_k, b_k \in [0, 1] \tag{2.4}$$

Different heuristic methods have been proposed and some of them are reported in Appendix A. As a result of the discounting procedure, a non-zero mass probability, D, for all *unseen events* is obtained:

$$D = 1 - \sum_k \hat{P}(c_k) \tag{2.5}$$

Then the probability, D, has to be redistributed among all unseen events, K_0. If we assume all unseen events are equally likely, the probability estimate of each unseen event, u_k, would be $\hat{P}(u_k) = D/K_0$. However, in most cases a uniform distribution over all unseen events might be too simplistic an approach. Two general procedures for assigning a probability to each of the *unseen events* have been proposed in the literature of speech recognition. The interpolation technique was first used in the past (Jelinek & Mercer, 1980). The principle behind interpolation is that the probability of a generic event W (seen or unseen) is a linear combination of the probabilities of all events that include W: for example, the event represented by the word pair (I WANT) includes the event (I WANT TO), (I WANT TO GO), etc. In the case of an n-gram language model, the n-gram probability $\hat{P}(w_n|w_1, \ldots, w_{n-1})$ is obtained by linear interpolation among the probabilities of events of lower-order n-gram models:

$$\begin{aligned} \tilde{P}_I(w_i|w_1, \ldots, w_{i-1}) = {} & \lambda_{i-1} \hat{P}_{ML}(w_i|w_1, \ldots, w_{i-1}) \\ & + \lambda_{i-2} \hat{P}_{ML}(w_i|w_2, \ldots, w_{i-1}) + \ldots + \lambda_0 \hat{P}_{ML}(w_i) \end{aligned} \tag{2.6}$$

where the interpolation parameters $\lambda_0, \ldots, \lambda_{i-1}$ are estimated using a corpus of sentences different from those used for estimating the original n-gram probabilities. However, the so-called *backoff* method was proved to be less burdensome as far as the computation of free parameters, λ_i, while giving good results (Jelinek & Mercer, 1980; Katz, 1987). The computation schema inferred by Equation (2.6) uses all probabilities of all events including W, while the *backoff* method, outlined by Katz (1987), takes into account only one observed event, namely the *closest* to W. In terms of the n-gram language model, the *closest* event to the word tuple $(w_i, \ldots, w_j)$ is the tuple $(w_{i+1}, \ldots, w_j)$. Thus, the *backoff* algorithm for an n-gram language model can be given in a recursive form:

$$\begin{aligned} \tilde{P}_B(w_i|w_1, \ldots, w_{i-1}) = {} & \hat{P}(w_i|w_1, \ldots, w_{i-1}) \\ & + \theta(\hat{P}(w_i|w_1, \ldots, w_{i-1})) \alpha(w_1, \ldots, w_{i-1}) \tilde{P}_B(w_i|w_2, \ldots, w_{i-1}) \end{aligned} \tag{2.7}$$

[1] In this work, unless otherwise specified, we assume that all the words in the unseen tuple have been observed in the training corpus.

where $\hat{P}(w_i|w_1, \ldots, w_{i-1})$ is a discounted estimate [see Equation (2.4)]; the $\theta(x)$ function is defined by:

$$\theta(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

and the coeffient of $\alpha(w_1, \ldots, w_{i-1})$ is a normalizing constant such that:

$$\sum_{w} \tilde{P}(w_i = w|w_1, \ldots, w_{i-1}) = 1 \tag{2.9}$$

where the sum is calculated for all words w observed in the training corpus.

The recursive procedure in Equation (2.7) is very efficient in terms of the required number of parameters with respect to the schema in Equation (2.6). On the other hand, the interpolation schema is a suitable approach to smooth the *n*-gram probability with other syntactic–semantic constraints (e.g. word classes).

3. Towards a unified representation of an *n*-gram language model

The purpose of this work is to show how an *n*-gram language model can be designed on the basis of a *unified* formalism, that is the Stochastic Finite State Automaton (SFSA). In this framework, the probability $\hat{P}(W)$ is derived as the probability of the word sequence W recognized by the SFSA and to each word sequences one or more sequences of states are associated. The motivation to build a stochastic automaton to realize the language model is twofold:

(1) To make the computation of $P(W)$ independent of the language model as long as it can be represented through a stochastic finite state automaton. The probability $P(W)$ of the string W (recognized by the automaton) is derived from the probabilities of the corresponding state sequences.
(2) To improve the efficiency of any search algorithm that is based on a network search (e.g. Viterbi's algorithm). In particular, speech recognition is performed by searching for the word sequence that maximizes a MAP criterion [Equation (1.2)]. With our approach, this process is formalized as the search of the "best" path in a network. Therefore, we can increase the decoder efficiency by compiling into the finite state network additional constraints (e.g. tree lexicon structured network, network with crossword constraints, etc.).

The difficulties arising in representing an *n*-gram language model through a stochastic finite state network are the following.

(1) The straightforward full network representation, where each probability $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$ is associated to an arc, is not possible, because it requires a number of arcs proportional to $|V|^n$, where $|V|$ is the number of words in the vocabulary V.
(2) The recursive schema of Equation (2.7) must be adopted in the finite state formalism to achieve efficient implementation of the backoff mechanism.

Another important issue is the design of word and phrase class-based language models:

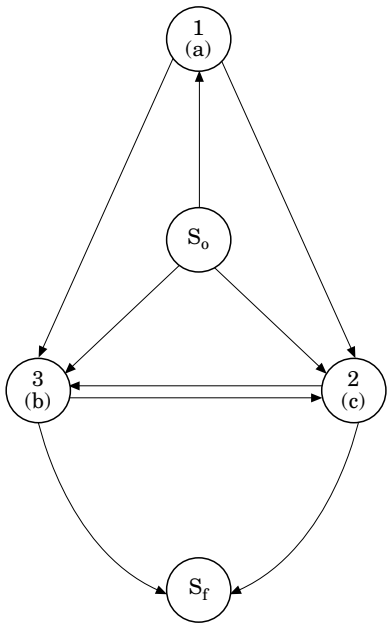

Figure 1. Example of a bigram network.

training algorithms must be designed to incorporate syntactic–semantic knowledge into a stochastic finite state automaton.

The idea of using a finite state stochastic automaton as a language model for speech recognition had been proposed by Jelinek (1976) and first applied to the Raleigh artificial language. More recently, a stochastic automaton network for bigrams has been applied to learn a grammar for large vocabulary speech recognition (Placeway, Schwartz, Fung & Nguyen, 1993). In our work, we present a general model for designing efficient and effective stochastic finite state automata for n-grams ($n \geqslant 1$) and class-based language models.

In the following paragraphs of this section, we introduce the simple case of the approximation of a bigram language model, through a finite state automaton. In Section 4.1, a general framework is presented: the VNSA (Riccardi, Bocchieri & Pieraccini, 1995).

3.1. Stochastic automata for bigram language models

Backoff strategies can be implemented in a stochastic finite state network in such a way that the speech recognizer does not have to handle the estimation formulas [Equation (2.7)] directly, but it rather has to perform a search through a network. In particular, we want to compute the probability $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$ based on $P(w_i|w_{i-k+1}, \ldots, w_{i-1})$ ($k<n$) according to *if-then-else* conditions like those in Equation (2.8). Assume we have a bigram network like the one shown in Fig. 1. The states s_0 and s_f are the initial and final states of the automaton. For the sake of clarity the transition probabilities are not shown. The vocabulary is composed of three words (a, b and c) and a state is associated to each word. Let us assume that only the events ab,

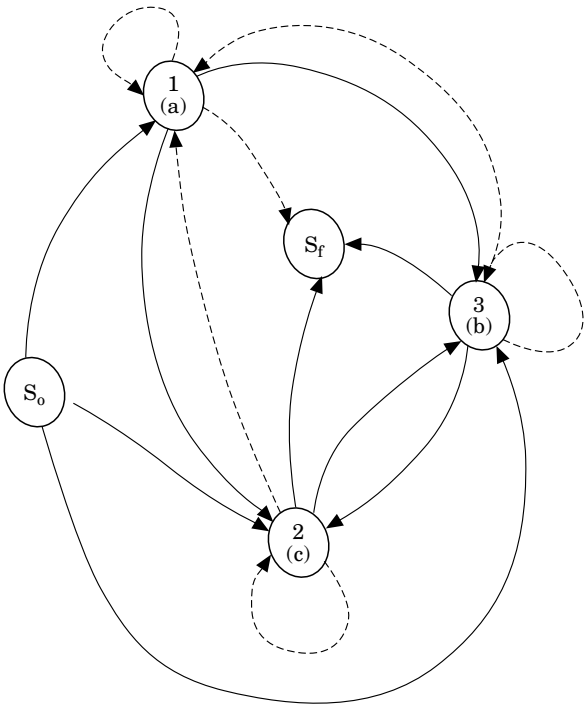

Figure 2. Example of a full bigram network.

ac, *bc*, *cb* have been observed and then we can compute the ML estimates $\hat{P}_{ML}(b|a)$, $\hat{P}_{ML}(c|a)$, $\hat{P}_{ML}(b|c)$ and $\hat{P}_{ML}(c|b)$. Using this network for parsing (recognizing) a sequence of words will fail when the sequence contains bigrams that were not observed, like in the sequence *aabacb*.

A solution is to add *all* the missing transitions (dashed arcs), as shown in Fig. 2, and to compute their probabilities according to the backoff estimation technique. The application of the *backoff* procedure results in the following estimates.

$$\begin{array}{lll} \tilde{P}(a|a)=\alpha_a\hat{P}(a) & \tilde{P}(b|a)=\hat{P}(b|a) & \tilde{P}(c|a)=\hat{P}(c|a) \\ \tilde{P}(a|b)=\alpha_b\hat{P}(a) & \tilde{P}(b|b)=\alpha_b\hat{P}(b) & \tilde{P}(c|b)=\hat{P}(c|b) \\ \tilde{P}(a|c)=\alpha_c\hat{P}(a) & \tilde{P}(b|c)=\hat{P}(b|c) & \tilde{P}(c|c)=\alpha_c\hat{P}(c) \end{array} \tag{3.1}$$

where the same notation as in Equation (2.7) is adopted. The *full n-gram* network (Fig. 2) cannot be realized in practice since the total number of transitions (i.e. N_V^n, where N_V is the size of vocabulary V) is too large for the implementation of the network in a real application. With a vocabulary of 1000 words, a trigram language model would

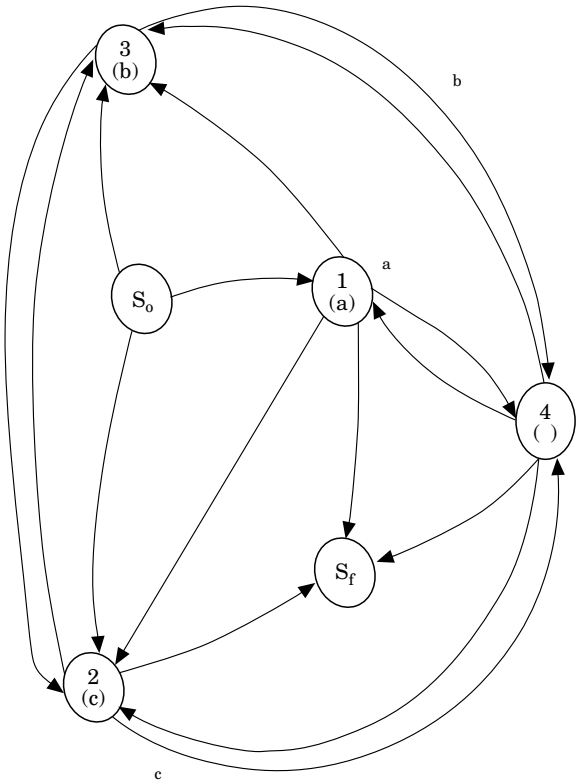

Figure 3. Example of a bigram network that approximates a full bigram network.

need to store one billion state transitions. The backoff probabilities defined by Equation (3.1) can instead be used within the network shown in Fig. 3.

It should be noticed that there is only one transition out of each node (arcs with associated probability α_a, α_b and α_c) to represent all the missing transitions. All those transitions are leading to a common *null* state (i.e. a state that does not recognize any word) that is represented by the symbol ε. The null node is then connected to each state in the network. The network described in Fig. 3 corresponds to the one proposed by Placeway *et al*. (1993) for bigram language modeling.

3.2. Non-determinism in stochastic automata

In the standard *n*-gram language model (Section 2) a word sequence W is assigned with a unique set of conditional probabilities $\tilde{P}(w_i|w_{i-n+1}, \ldots, w_{i-1})$ and the probability $P(W)$ is computed through Equation (2.1). When a language model is represented through a stochastic finite state automaton, an input word sequence W can be accepted by the automaton through one or more state sequences (paths). In particular, the automaton that allows for multiple paths is called non-deterministic. For instance, in the previous example, the sentence **abbc** can be recognized through the sequences of states **ab**∈**bc**, **a**∈**b**∈**bc**, **ab**∈**b**∈**c**, etc. Of course, the path that gives the probability

corresponding to the backoff [Equation (2.7)] is **ab**∈**bc**. However, when the stochastic automaton is used within the Viterbi algorithm only the path that maximizes Equation (1.2) will be chosen. The probability of a sentence W given by the Viterbi algorithm can be written as:

$$P_{vit}(W) = P(W, \hat{\xi}) \tag{3.2}$$

where:

$$\hat{\xi} = \arg\max_{\xi \in \Xi_W} P(W, \xi) \tag{3.3}$$

being Ξ_W the set of all sequence of states ξ that account for the word sequence W. The probability of W is then:

$$P(W) = \sum_{\xi \in \Xi_W} P(W, \xi) \tag{3.4}$$

If the automaton is deterministic there is only one sequence of states that accounts for the input words; hence, we can write:

$$P(W) = P_{vit}(W) \tag{3.5}$$

If the automaton is non-deterministic we generally make the assumption that the maximum term is prevailing in the sum of Equation (3.4), hence:

$$P(W) \approx P_{vit}(W) \tag{3.6}$$

but we cannot always assume that this hypothesis is true. However, we can compute the word sequence probability either by means of the *forward* algorithm (Baum, 1972) or by using its Viterbi approximation. The difference between the two values we define to be the *degree* of the network non-determinism. The comparison of Equations (2.1) and (3.4) points out the differences between the standard n-gram language model and the stochastic automaton. In the latter case, the probability $P(W)$ is defined in terms of paths and transition probabilities. Both the sequences of states and the transition probabilities are determined by the model used to build the automaton. In the following section, we propose our approach to build stochastic automata for language modeling: the Variable N-gram Stochastic Automaton (VNSA). In Section 7 we evaluate the goodness of the VNSA as an approximation to the standard n-gram models.

4. The Variable *N*-gram Stochastic Automaton (VNSA)

The design of a stochastic automaton is described by the state transition function and the state transition probabilities. Our algorithm defines the state transition function based on the set of events (word tuples) occurring in the training set $\mathcal{T}$ and associates these events to the states. In Section 4.1 we present the VNSA and the model description is completed in Section 4.2 with the algorithms used to compute the state transition probabilities.

4.1. Definition of the Variable N-gram Stochastic Automaton (VNSA)

A stochastic non-deterministic finite state automaton $\mathcal{Q}$ is described by the quintuple $\{\mathbf{S}, \mathbf{V}, \mathcal{F}, s_0, \mathbf{S}_f\}$, where $\mathbf{S}$ is a set of states s, $\mathbf{V}$ is a set of words (including the empty word ε), s_0 is the initial state and $\mathbf{S}_f \subset \mathbf{S}$ is a set of final states. $\mathcal{F}$ is a state transition function that, given a state s and an input word $w \in \mathbf{V}$, returns the set of pairs $\{(t_k^w, p_k^w)\}$: $\mathcal{F}(s, w) = \{(t_k^w, p_k^w)\}$ $(k \geqslant 1)$, where p_k^w are the transition probabilities of going from state s to state t_k^w, for a given input word w. In particular, the automaton is non-deterministic since we allow $\mathcal{F}(s, w)$ to be a one-to-many mapping. The recognition process of an input word sequence $W = w_1, w_2, \ldots, w_N$ starts with the application of $\mathcal{F}$ to the initial state s_0 (instant 0) and input word w_1. Then, when in state s at the generic instant i and given an input sequence word $w_i \in \mathbf{V}$, the automaton will:

(1) apply state transition function to the state s with input word $w_i (\neq \varepsilon)$, move to the corresponding next state (if $\mathcal{F}(s, w_i) \neq \emptyset$), and read the next word in the input string, and
(2) apply the state transition function to the state s with the null word ε, and move to the corresponding next state (the current input word is not changed).

In the following, a state reached only through a null word will be called null state.

A VNSA is a non-deterministic stochastic finite state automaton. Each state $s \in \mathbf{S}$ is associated with an m-tuple observed in the training set, $v_1, \ldots, v_m$ with $0 \leqslant m < n$ (n is called the order of the automaton). The m-tuple $v_1, \ldots, v_m$ is called the *history* of the state s and m is called the history size. In the case $m = 0$ the empty word is associated to s. The history $v_1, \ldots, v_m$ of a generic state s is considered only for the purpose of the design and training of the VNSA and it is not used while parsing word sequences.

At the generic instant i, given the current word w_i and the current state s with history $v_1, \ldots, v_m$, only two types of transition are defined by the state transition function. Type 1 transition consumes an input word w and moves either to the non-null state t_1^w with history $v_1, \ldots, v_m, w$ (the history size is increased), or to the non-null state t_2^w with history $v_2, \ldots, v_m, w$ (the history size does not change). Type 2 transition does not consume any input symbol, and moves to a null state t^ε (with history $v_2, \ldots, v_m$) with *backoff* probability p^ε: this transition implements the *backoff* mechanism (the history size is decreased) and it corresponds to a loss of part of the current state history in order to represent the next state in the automaton. The transition of type 2 is always available, while the transition of type 1 may not be. It must be observed that in this kind of automaton all transitions leading to a non-null state with history $v_1, \ldots, v_m$ recognize word v_m, while transitions leading to a null state always recognize the empty word ε. The predecessor s of a null state with history $v_1, \ldots, v_m$, has history of the kind $v, v_1, \ldots, v_m$ ($v \in \mathbf{V}$ and $v \neq \varepsilon$). Moreover, *only* a transition to the null state with history $v_1, \ldots, v_m$ is needed to decrease by one the history size of the state s with history $v, v_1, \ldots, v_m$. As a result, for a state s with history $v_1, \ldots, v_m$, m transitions to null states are necessary to lower down to zero the history size.

The automaton network, defined above, will be referred in the following as the canonical realization of the VNSA.

In Fig. 4 a portion of a third-order VNSA network is shown. The vocabulary of the automaton is $V = \{a, b, c, d, \varepsilon\}$. For the sake of clarity transition probabilities are not shown and the symbols activating a transition are on the arcs. Null states have a rectangular shape and non-null states have a round shape. The history of a state s is

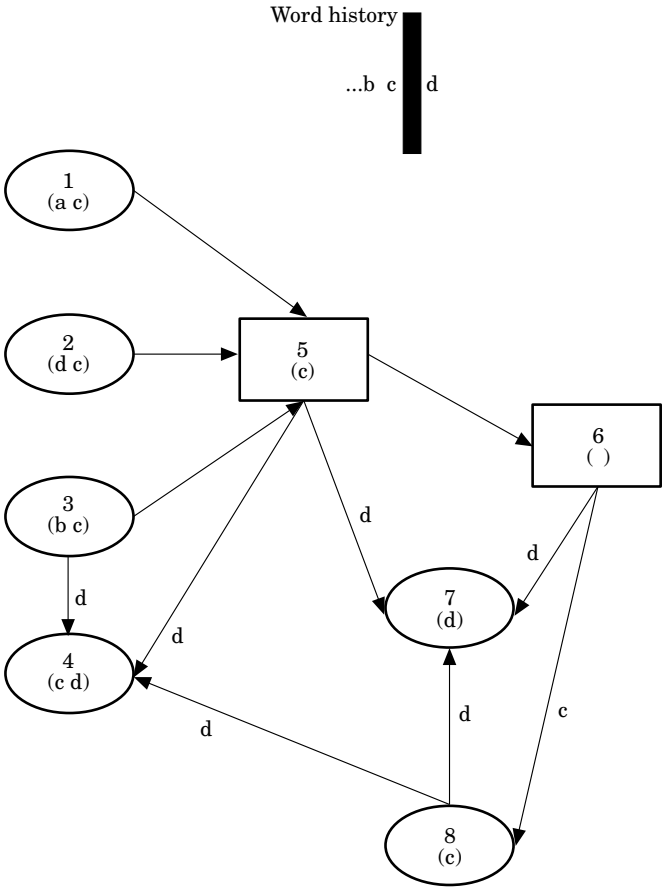

Figure 4. Portion of third-order VNSA network.

shown within parentheses in order to illustrate the relationship between the event in the training set associated to s and the available transitions. Let us assume that the automaton is processing symbol d, while the last two symbols read are b and c. The automaton might be in state 3 or in state 8 since c was the symbol previously processed. From state 3 the automaton can step into state 4 or into state 5. State 5 is a null state and state 1 (history ac), 2 (history dc) and 3 (history bc) can decrease the history size through the same null state 5 with distinct *backoff* probabilities. Once in state 5 the automaton can release an additional history symbol by going into null state 6, or increase the history size by going into state 4, or keep the same history size by stepping into state 7. From state 8 the automaton can either keep the same history size by going into state 7 or increase it by stepping into state 4. From the analysis of Fig. 4, it is clear how the non-determinism of the VNSA has been designed to cope with the need of efficiency (shared transition to null states) and flexibility (variable length word contexts).

4.2. Estimation of probabilities for the VNSA

For each state s (with history $v_1, \ldots, v_m$) of the VNSA automaton there is a set of words $w \in \mathbf{w}_s$ ($w \neq \varepsilon$) such that $\mathscr{F}(s, w) \neq 0$ and $\mathscr{F}(s, w) = \{(t_1^w, p_1^w), (t_2^w, p_2^w)\}$. For a state s the following equation holds:

$$\sum_{w \in \mathbf{w}_s} (p_1^w + p_2^w) + p^\varepsilon = 1 \tag{4.1}$$

where $p_1^w (p_2^w)$ is the conditional probability $\hat{P}(t_1^w|s)(\hat{P}(t_2^w|s))$ and the word w is the input symbol (transition of type 1). The probability p^ε is the probability of decreasing the history size of state s by one (transition of type 2). The similarity of Equations (4.1) and (2.5) and the considerations in Section 2 let us compute the probability of loss of memory, p^ε (*backoff* probability), with one of the methods described in Appendix A. The probability $\hat{P}(w|v_1, \ldots, v_m)$ is redistributed among p_1^w and p_2^w according to an interpolation schema. In particular, the probability $p_1^w (p_2^w)$ corresponds to the transition from the state s to the state $t_1^w (t_2^w)$ with associated word tuple $v_1, \ldots, v_m, w$ ($v_2, \ldots, v_m, w$). Hence, the two transitions from s to t_1^w and t_2^w correspond to the same event in the training set, that is the word tuple $v_1, \ldots, v_m, w$, and p_1^w and p_2^w are implicitly defined by the following estimate:

$$\begin{aligned} \hat{P}(w|v_1, \ldots, v_m) &= \beta \hat{P}(w|v_1, \ldots, v_m) + (1-\beta)\hat{P}(w|v_1, \ldots, v_m) \\ &= \hat{P}(t_1^w|s) + \hat{P}(t_2^w|s) \qquad (4.2) \\ &= p_1^w + p_2^w \qquad (4.3) \end{aligned}$$

The free parameter β is calculated within a cross-validation framework as described in Section 4.3.

Given the non-determinism of the VNSA, a word sequence $W = w_1, w_2, \ldots, w_N$ can be recognized along multiple state sequence ξ (path). Each path ξ corresponds to a different decomposition of the probability $\hat{P}(W)$ into n-gram type probabilities. In fact, each transition probability from a state s to a state t is computed either as an n-gram type probability [see Equation (4.3)] or as a *backoff* probability. Along each single path, the probability $\hat{P}(W, \xi)$ is computed through a variable length n-gram model:

$$\hat{P}(W, \xi) = \prod_i \alpha_i \beta_i \hat{P}(w_i|w_{i-n_{i,\xi}}, \ldots, w_{i-1}) \tag{4.4}$$

where α_i are *backoff* probabilities, $\hat{P}(w_i|w_{i-n_{i,\xi}}, \ldots, w_{i-1})$ are discounted probabilities, β_i are defined as in Equation (4.3) and each path ξ has associated an index sequence $n_{1,\xi}, \ldots, n_{N,\xi}$, Then the probability estimate $\hat{P}(W)$ can be written as:

$$\hat{P}(W) = \sum_{\xi \in \Xi_W} \hat{P}(W, \xi) = \sum_{\xi \in \Xi_W} \prod_{(s,t) \in \xi} \hat{P}(t|s) \tag{4.5}$$

where Ξ_W is the set of all available paths ξ to parse the word sequence W and the pair (s, t) is contained in the state sequence ξ. While all paths in Ξ_W have to be taken into

account to compute the probability $\hat{P}(W)$ (see Section 3.2), the Viterbi algorithm selects the path which gives the best performance in the global maximization of probability $P(W|O)$ [see Equation (1.2)]. Equations (4.4) and (4.5) show how the VNSA's variable length history paradigm and non-determinism are exploited to both implement the backoff mechanism to recognize the unseen word sequence in the training set $\mathscr{T}$ and to smooth n-gram probabilities in the spirit of the interpolation schema of Equation (2.6).

4.3. Estimation of free parameters

The estimation of the *backoff* probabilities α_i can be thought of as the estimation of free parameters as in the case of the linear discount method: $a_k = \alpha$ and $b_k = 0\ \forall k$ in Equations (2.4) and (2.5). Furthermore, the parameter β in Equation (4.3) is a free parameter for the transition probabilities of the VNSA automaton network. In both cases an estimated probability $\hat{P}$ is decomposed through a convex sum into two terms $\hat{P}_1$ and $\hat{P}_2$:

$$\hat{P} = \gamma\hat{P} + (1-\gamma)\hat{P} = \hat{P}_1 + \hat{P}_2 \tag{4.6}$$

where γ $(0 \leqslant \gamma \leqslant 1)$ is the generic free parameter. For the estimation of the free parameters we have used a similar version of the cross-validation method as described by Stone (1974). The performance of a stochastic language model is usually assessed by estimating its perplexity (Jelinek & Mercer, 1980) which is a function of the estimated entropy of the language whose source is assumed to be ergodic. The estimated entropy is then defined as:

$$\hat{E} = -\frac{1}{n}\log \hat{P}(w_1, \ldots, w_n) \tag{4.7}$$

and the perplexity P_E is defined as:

$$P_E = 2^{\hat{E}} \tag{4.8}$$

Thus, the computation of the free parameters can be performed by minimizing $\hat{E}$ on a test set and estimate the n-gram probabilities $\hat{P}(w_i|w_{i-n+1}, \ldots, w_{i-1})$ on a training corpus. However, the estimation of the free parameters can be carried out in such a way that each sample in the corpus $\mathscr{T}$ is used to evaluate $\hat{E}$ and for this purpose we have partitioned the corpus $\mathscr{T}$ in $\mathscr{N}$ subsets $\mathscr{T}^i$ such that $\mathscr{T} = \cup_i \mathscr{T}^i$ and $\mathscr{T}^i \cap \mathscr{T}^j = \emptyset$. The set $\mathscr{T}^{li} = \cup_{j \neq i} \mathscr{T}^j$ is used to train the stochastic model for a fixed γ value and the test set entropy (cost function) $\hat{E}_{\mathscr{T}^i}(\gamma)$ is calculated on the *held out set* $\mathscr{T}^i$. In the case of one free parameter γ the test set entropy $\hat{E}_{\mathscr{T}^i}(\gamma)$ can be written as:

$$\hat{E}_{\mathscr{T}^i}(\gamma) = \texttt{const} - \frac{1}{n}\left(\sum_i^n c_i \log(\gamma) + \log(-a_i\gamma + b_i)\right) \tag{4.10}$$

where n is the number of words in the test set and the coefficients a_i, b_i and c_i $(\in[0, 1])$ depend on the estimates $\hat{P}(w_i|w_{i-n+1}, \ldots, w_{i-1})$. As a result of setting the first derivative

of $\hat{E}_{\mathscr{T}^i}(\gamma)$ to zero in Equation (4.10), a local minimum $\hat{E}_{\mathscr{T}^i}(\gamma_m)$ is guaranteed. This process is repeated $\mathscr{N}$ times, so that each subset $\mathscr{T}^i$ is employed to estimate γ. Then we have performed the optimization of the parameter γ on all the possible *held out sets*:

$$\gamma_{opt} = \arg\min_{\gamma} \sum_{i} \hat{E}_{\mathscr{T}^i}(\gamma) \tag{4.11}$$

The same reasoning can be extended for multiple free parameters γ_i obeying the estimation formula [Equation (4.6)]. This approach has been used for computing the parameters in the *linear* discount method (see Appendix A) and for the transition probabilities in the VNSA automaton.

4.4. Structure of the VNSA network

In this section we describe the overall structure of the VNSA network. In Fig. 5 a portion of the VNSA network is shown. Four network partitions, A, B, C and D, have been pointed out. They contain states with history size (*hs*) 2, 1, 1 and 0, respectively.

Section D consists of just one null state with zero history size. Network sections A and C have non-null states connected directly among them, within the partition. More precisely, for a state sequence ξ containing states all belonging to either partition A or C, in Equation (4.4) we have that $n_i = n$ ($n = 2$ or 1) and $\alpha_i = 1{\cdot}0\ \forall_i$.

Partition B contains only null states not connected among them but to states belonging to partitions A and C. In particular, for a state sequence ξ containing a state in B, in Equation (4.4) we have an index i such that $0 < \alpha_i < 1{\cdot}0$ and $n_i < n_{i-1}$. The transition probability from a non-null state in A to a null state in B is the *backoff* probability p^{ε}.

In the most general case, from a non-null state with history size *hs* ($hs \neq 0$) it can be reached a state with history size of either $hs+1$ (e.g. from a state in C to one in A), *hs* (e.g. a transition within partition A or C) or $hs-1$ (e.g. from a state in A to one in B). As a whole, the automaton from a non-null state with history length $hs \geqslant 1$ can reach a null state with zero history size without processing any new input symbol. On the other hand, the history size can only be increased by one at each new input word.

The number of parameters of the *canonical* VNSA network depends linearly on the training data size. The number of states and transitions in section C (M_C and T_C) is a function of the number of different words and pairs, respectively, observed in the training data. The number of states and transitions in partition A (M_A and T_A) depends on the number of pairs and triples, respectively. In general, if a partition contains states with history size *hs*, then the partition has a number of states and transitions depending on the number of *hs*-tuples and $hs+1$-tuples, respectively.

4.5. Heuristic state minimization of the canonical VNSA network

In this section we deal with the issue of reducing the total number of parameters (states and transitions) needed to build the *canonical* VNSA network. The sensitiveness of the VNSA *canonical* network performance, with respect to the state reduction procedure, is evaluated in terms of perplexity and, more significantly, in terms of word error rate. The principle used to prune states in the VNSA network is based on thresholding the number of occurrences of the word contexts corresponding to each state. The algorithm

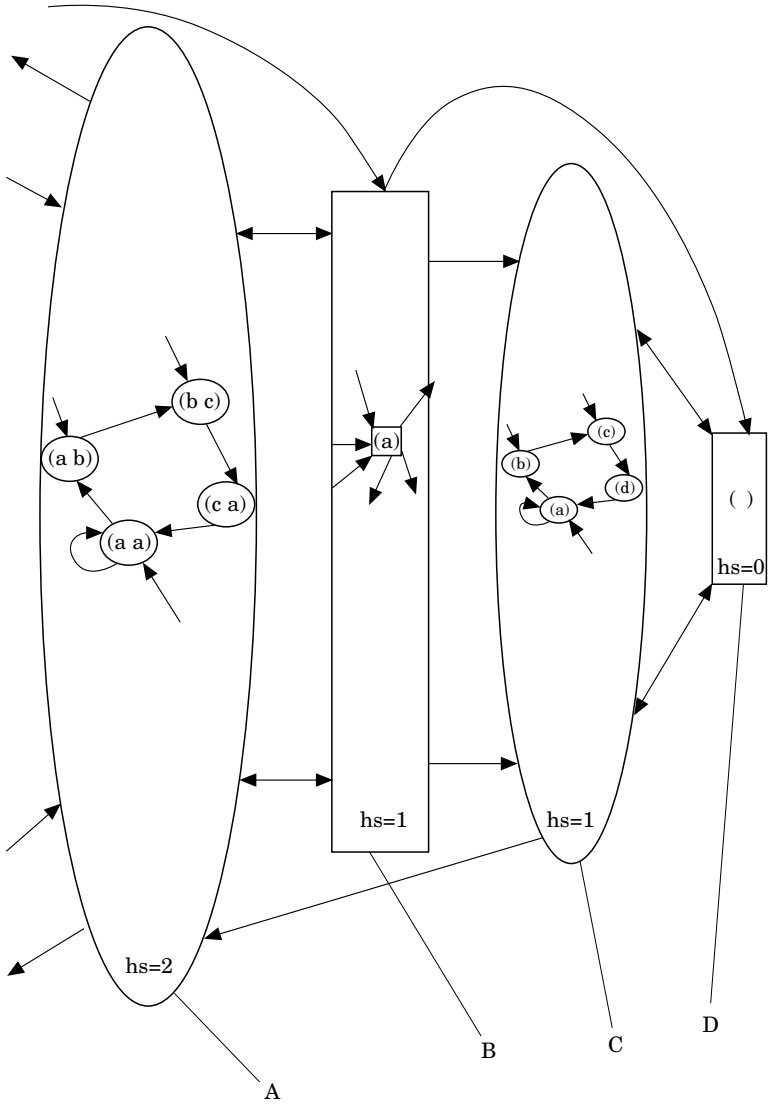

Figure 5. Overall structure of the VNSA network.

modifies the *canonical* VNSA network realization and decreases the number of states from M_i to M_f and the transitions from T_i to T_f until a given reduction rate $\tau = \tau(M_f/M_i, T_f/T_i)$ is achieved. The following pseudo code describes the pruning procedure.

(1) `i = 1`
(2) Search for `the` states having a corresponding word context occurring `i` times.
(3) Redistribute the transition probabilities of `the` predecessors of `these` states and delete `them`.
(4) `if` the reduction rate is greater than τ
   - `then` increment `i` by one and go to 2.
   - `otherwise` algorithm ends.

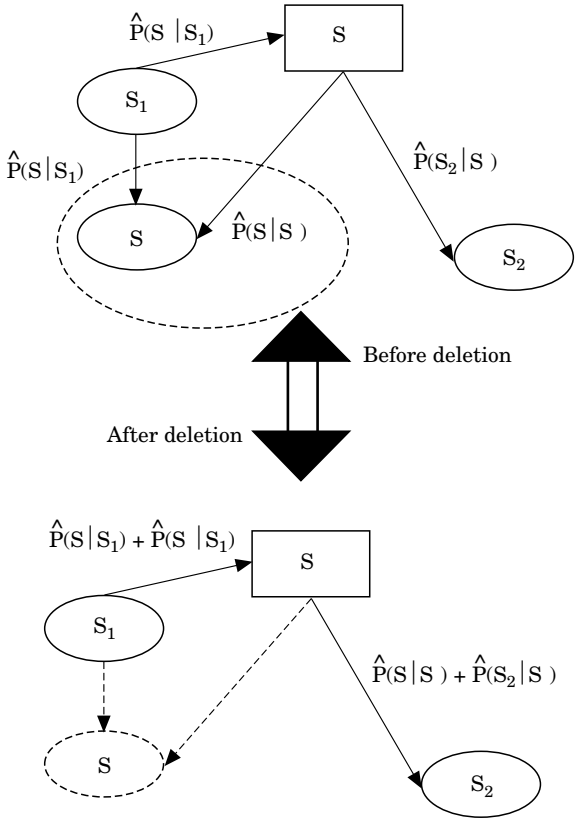

Figure 6. Probability distribution schema in the state pruning algorithm.

The key step of the algorithm is the redistribution of the probabilities and we describe that in Fig. 6. Let us suppose that the state s has to be deleted from the network. If one of the preceding states is a non-null state s_1, then the transition probability $\hat{P}(s|s_1)$ is added to the *backoff* probability $\hat{P}(s_\varepsilon|s_1)$, as pointed out in Fig. 6. If the preceding state is a null state, s_ε, the transition probability $\hat{P}(s|s_\varepsilon)$ is added either to the probability of going into non-null state s_2 (if it is available, as in Fig. 6) or to the *backoff* probability of state s_ε (not shown in Fig. 6).

This heuristic state minimization has given good results and for a 50% reduction of the VNSA's parameter number, we have obtained *only* a 0·1% increase of the word error rate. More details on the size and perplexity of the networks are given in Section 7. As a final remark, it is worthwhile noting that the cross-validation framework described in Section 4.3 can also be applied to minimize the perplexity for a given reduction rate τ.

5. Relationship between perplexity, word error rate and non-determinism

We have performed experiments to investigate the relationship between the perplexity, the word error rate of the speech recognizer and the non-determinism introduced by the VNSA automata. To this purpose, the VNSA network approximating a bigram

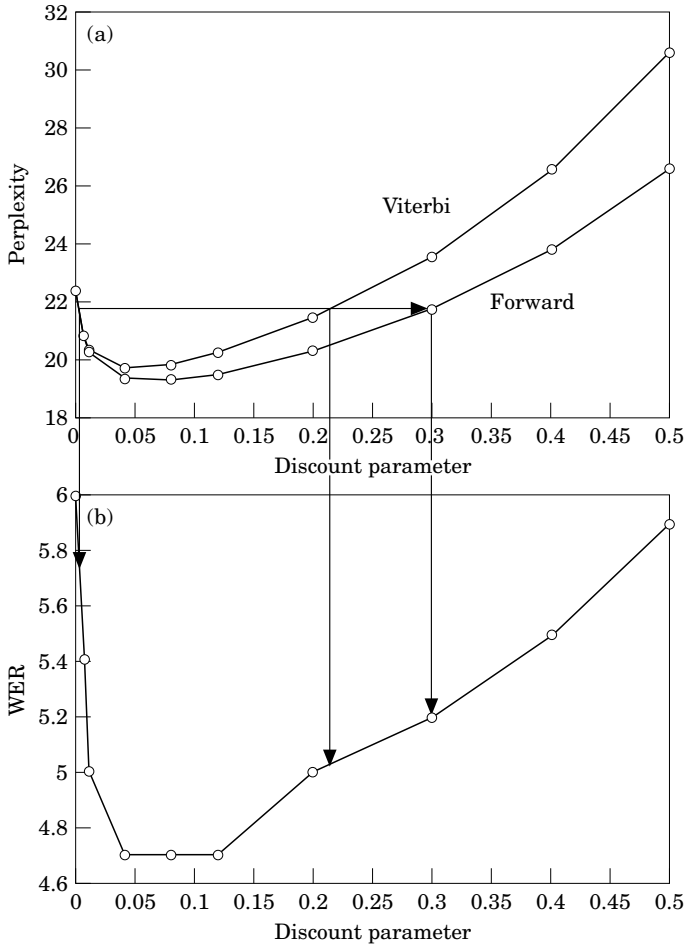

Figure 7. (a) Test set perplexity (Viterbi and Forward computation) and (b) word error rate (WER) vs. discount parameter α.

language model (see Fig. 3) has been used. The training set consists of 20 844 sentences from the ATIS corpus[2] corresponding to a total of 208 103 words. The test set (i.e. ATIS official December 1994 test set) consisted of 981 sentences corresponding to 10 081 words. The size of the vocabulary is $N_V = 1530$ words. The transition probabilities were computed with the *uniform linear* method (see Appendix A) which allows for continuous variation of the test perplexity. In particular, it has been shown in Section 4.3 that a local minimum is guaranteed. The continuous variation of the parameter α corresponds to a continuous variation of the language model performance (i.e. in terms of perplexity). The speech recognizer used in all experiments presented in this work is the one designed for the 1994 ATIS evaluation and it is described by Bocchieri, Riccardi and Anantharaman (1995).

Fig. 7(a) shows the perplexity behavior for different values of the discounting

[2] The Air Travel Information System (ATIS) corpus is a set of questions about flight information, fares, etc. The speech database is spontaneous and it is collected through a Wizard-of-Oz paradigm.

parameter α. In order to compute the probability $\hat{P}(w_1, \ldots, w_n)$ in Equation (4.7) we have used the Viterbi approximation [Equation (3.6)] and the forward algorithm. The difference between the two values (degree of non-determinism) increases as the value of α is increased (see Section 3.2). Besides, in the interval centered around the local minimum point α_{min} for small variations of the parameter α we have that $P_E \simeq P_E(\alpha_{min})$ [see Equation (4.10)]. In Fig. 7(b) the graph of the word error rate is plotted. The same behavior of the perplexity curves of Fig. 7(a) is found in the WER plot in Fig. 7(b) and in particular for a given value of the perplexity, the WER value is lower where the degree of non-determinism is higher (right side of the curve). From these experiments we have seen that, even though non-determinism in VNSAs is paid in terms of computational load for the Viterbi decoding, it is critical to achieve the lowest error rates as well as to make the system less sensitive to variation of the language model performance.

6. Word and selected-phrase classes

6.1. Benefit of using word and selected-phrase classes

The generalization capability of the language model can be greatly achieved by integrating semantic–syntactic knowledge to estimate the word sequence probabilities. In particular, in this paragraph we discuss the use of word classes (e.g. *city* = {BOSTON, CHICAGO, . . .}, *number* = {ONE, TWO, . . .}) and selected-phrase classes (e.g. I WOULD LIKE TO, WHAT KIND OF, . . .). The use of word and selected-phrase classes are intended here as a coarse but effective linguistic and statistical classification of the word sequences in the training corpus. As a result, this leads to a language model which integrates high-level knowledge into the speech recognition processes. There are four main reasons why word classes and selected-phrases should be helpful. First, in a stochastic framework, the probability estimates are more robust against data sparseness. Secondly, syntactic–semantic knowledge can be shared among language models pertaining to the training corpora of similar tasks. For instance, word (e.g. *city*, *number*, *airport*, etc.) and selected-phrase classes (e.g. request phrase, specification phrases, etc.) are common to any application involving those concepts. Thirdly, class-based language models are efficient in terms of memory storage. Last, but not least, the speech recognition process benefits (in terms of accuracy) from the use of word classes and selected-phrases (Bocchieri, Riccardi & Anantharamus , 1995).

6.2. Application to the VNSA automaton

The use of the word and selected-phrase classes within the VNSA paradigm is quite straightforward. The training corpus is tagged with word and selected-phrase class labels so that each word or word sequence is replaced with the corresponding label. Then a VNSA automaton is built on the basis of the labeled training set. At this stage, the automaton is able to recognize all possible sequences of class labels. The computation of the estimate $\hat{P}(W)$ can be obtained after the expansion of each word and selected-phrase label (see Appendix B for details on the probability computation). As a result the application of the word classes and selected-phrases to the VNSA paradigm involves four steps as follows:

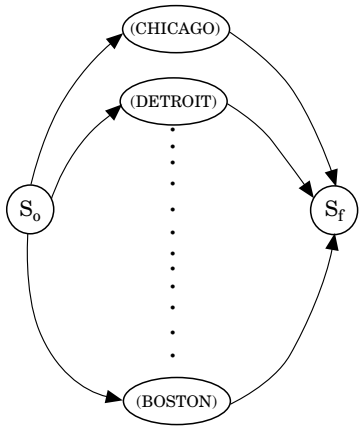

Figure 8. An example of word class automaton.

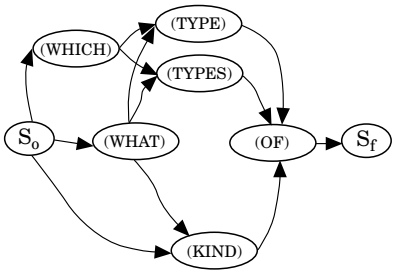

Figure 9. An example of phrase class automaton.

- **Lexical Analysis:** the sequence of words in the training corpus is tagged with the word and selected-phrase class labels. In Figs 8 and 9 are shown an example of word and phrase class automaton used to label the training corpus.
- **Stochastic Modeling:** a VNSA automaton is built out of the filtered training set.
- **Expansion:** all along the VNSA network obtained in the previous step, each label (corresponding to a word or phrase class) has to be expanded into an automaton as shown in Figs 8 and 9. Label expansion is performed in all states of the network obtained in the previous step.
- **Word and Selected-Phrase Class Automata Training:** in this final step all the transition probabilities inside the automata of the types shown in Figs 8 and 9 have to be estimated. As first assumption, a uniform distribution can be adopted for each state. However, Viterbi training has given better results in terms of perplexity and WER.

While estimating probabilities at the last step of the procedure, each class (word or selected-phrase class) might contain elements unseen in the training set: in this case, we use discount methods (see Appendix A) to compute the probability of a word w_i (not observed in the training set) given a word or a selected-phrase class $\mathscr{D}_i$.

In the literature, algorithms for automatic clustering of words have been studied (Brown *et al.*, 1992; Kneser & Ney, 1993; Pereira, Tishby & Lee, 1993), as well as for automatic computation of selected-phrases (Magermann & Marcus, 1990; Giachin,

TABLE I. Test set perplexity comparison between Katz's model and the VNSA

	Model order	
	2	3
VNSA	18·95	14·91
Katz	19·09	14·72

TABLE II. Test set perplexity for the *add-c* method

	VNSA order			
Set type	2	3	4	5
$\mathscr{T}$	18·95	14·91	13·92	14·01
$\mathscr{T}$*	17·44	12·17	12·24	12·40
$\mathscr{T}_f$	12·38	9·05	8·78	8·87

1995). As far as the application to the ATIS domain, we have built the word classes by hand (13) and we scored the whole set of selected-phrases in the ATIS training corpus based on the mutual information measure (Fano, 1961). Then we have identified 18 selected-phrase classes. As a whole, a total number of 31 classes have been used to design the class-based language models for the ATIS 1994 evaluation (Riccardi, Bocchieri & Pieraccini, 1995) and for the experiments presented in the next section.

7. Language model performances

One of our goals is to adopt the VNSA stochastic network as an approximation of the well-known *n*-gram language model into a one-pass Viterbi decoder. In order to compare the two language models we used the Katz bigram and trigram language model (Katz, 1987) and we evaluated the test perplexity on the ATIS data provided by the ARPA agency for the ATIS 1994 evaluation. From Table I we can draw the conclusion that the VNSA networks are equivalent in terms of perplexity to the *n*-gram model. In the following sections we present a performance analysis for different estimation methods and for increasing model orders. We also show that the VNSAs benefit from the integration of syntactic–semantic knowledge and it improves both the prediction power (perplexity) of the language model and the word accuracy of the speech recognition. We conclude this section with results from the heuristic minimization algorithm presented in Section 4.5 and with the WER scores for different VNSA models.

7.1. Test set perplexity for different probability estimation methods

The results presented below are given for two different types of test set, $\mathscr{T}$ and $\mathscr{T}_f$. The first one, $\mathscr{T}$, has been designed for the ATIS 1994 evaluation (the same used in Table I) and the second one, $\mathscr{T}_f$, is a filtered version of the first one. Precisely, the second type of training set is obtained as in the first step of the procedure presented in Section 6.2, by tagging $\mathscr{T}$ with the word and selected-phrase class labels. Tables II,

TABLE III. Test set perplexity for the *add*-1 method

	VNSA order			
Set type	2	3	4	5
$\mathscr{T}$	21·12	18·95	21·19	23·08
$\mathscr{T}_f$	13·39	11·16	11·72	12·36

TABLE IV. Test set perplexity for the *sub*-1 method

	VNSA order			
Set type	2	3	4	5
$\mathscr{T}$	19·45	15·62	15·92	16·34
$\mathscr{T}_f$	12·37	9·53	9·49	9·66

TABLE V. Test set perplexity for the *uniform linear* method

	VNSA order		
Set type	2 ($\alpha_{opt}=0{\cdot}08$)	3 ($\alpha_{opt}=0{\cdot}2$)	4 ($\alpha_{opt}=0{\cdot}3$)
$\mathscr{T}$	19·44	14·69	14·51
$\mathscr{T}_f$	12·68	9·37	9·08

III, IV and V report the perplexity scores, respectively, for VNSAs whose transition probabilities have been estimated through *add-c*, *add*-1, *sub*-1 and *uniform linear* discount methods (see Appendix A for details). For the *uniform linear* method the cross-validation method was used to optimize the discount value, α_{opt}. From the comparisons of these results the *add-c* method outperforms all the others and it is the method that has been used for application to speech recognition (Bocchieri, Riccardi & Anantharaman, 1995). Moreover, the transition probabilities estimated with the *uniform linear* method have a number of backoff probabilities dependent on the order of the VNSA automaton and the results in Table V are similar to those in Table II. In particular, this technique has been adopted for conceptual modeling with VNSA automata (Pieraccini & Levin, 1994). On the second line of Table II the perplexity scores are given for VNSA automata built when using word classes and selected-phrases (see the procedure described in Section 6.2). These are the overall best performances achieved with VNSAs and they support the benefit expected from the use of linguistic knowledge for language modeling. Moreover, even in terms of WER the class-based language model outperforms the Katz trigram language model and the canonical VNSA automaton built out of the non-filtered training data $\mathscr{T}$ (Riccardi, Bocchieri & Pieraccini, 1995).

TABLE VI. Development set perplexity of a third-order VNSA for different pruning thresholds

	tsh				
Set type	0	1	2	3	4
$\mathscr{T}^i$	16·02	16·17	16·34	16·55	16·79
$\mathscr{T}^i_f$	10·24	10·51	10·68	10·82	10·91

TABLE VII. Development set perplexity of a fourth-order VNSA for different pruning thresholds

	tsh				
Set type	0	1	2	3	4
$\mathscr{T}^i$	16·13	15·88	16·00	16·12	16·26
$\mathscr{T}^i_f$	10·08	10·10	10·13	10·19	10·26

TABLE VIII. State and transition number for the VNSAs in Table VI

	tsh				
	0	1	2	3	4
State number	19 877	11 842	9 339	8 113	7 388
Transition number	133 562	95 727	82 438	75 386	70 936

7.2. *Performance of pruned VNSA networks*

The procedure described in Section 4.5 has been applied within the cross-validation framework. Hence, the training set $\mathscr{T}(\mathscr{T}_f)$ has been split in two parts: the training set $\mathscr{T}^{li}(\mathscr{T}^{li}_f)$ and the held out part $\mathscr{T}^i(\mathscr{T}^i_f)$. The set $\mathscr{T}^{li}(\mathscr{T}^{li}_f)$ is used for training the VNSA automaton and $\mathscr{T}^i(\mathscr{T}^i_f)$ is used as the development set. Tables VI and VII report the experiments on a third- and fourth-order VNSA automaton, respectively. For the sake of clarity, in Tables VI and VII are shown the perplexities over the set $\mathscr{T}_i(\mathscr{T}^i_f)$ for one of the possible pair $(\mathscr{T}^{li}, \mathscr{T}^i)$ $((\mathscr{T}^{li}_f, \mathscr{T}^i_f))$.[3] The cutoff threshold *tsh* on the occurrence number of word contexts varies between 0 (no pruning is applied) and 4. Tables VIII

[3] The *add-c* method has been used to train the VNSA automaton on the set $\mathscr{T}^{li}$.

TABLE IX. State and transition number for the VNSAs in Table VII

	tsh				
	0	1	2	3	4
State number	81 538	53 259	46 459	43 401	41 641
Transition number	398 330	275 193	244 296	229 993	221 508

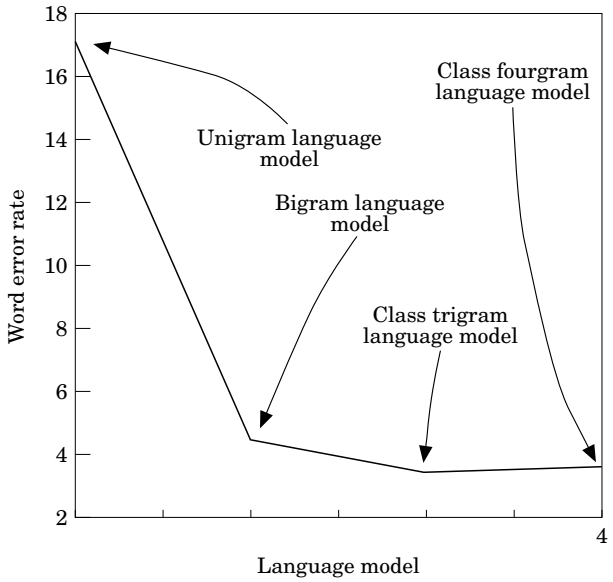

Figure 10. Word error rate vs. VNSA models.

and IX give the number of parameters for different values of the cutoff threshold *tsh*. From Tables VI and VIII it can be seen that for a third-order VNSA automaton, the perplexity increase is only 3% when the number of parameters is halved. As far as the accuracy performances, the use of networks with a halved parameter set allows for a WER increase of *only* 0·1% with respect to the *canonical* network realization.

7.3. WER performance as a function of the language model

In Fig. 10 the word error rate score as a function of the type of VNSA model is plotted. The leftmost WER value refers to the unigram language model used and the word error rate is 17%. Acceptable WER scores are reached only when bigrams are used. For the third- and fourth-order VNSA automaton word and phrase classes (as described in the previous section) were used. No improvement was obtained for the fourth-order VNSA automaton. As a final remark we point out that the recognition system achieves quite accurate scores with the second-order VNSA network, while the best system asset has to be obtained by means of a higher-order language model integrating adequate linguistic knowledge.

It is worth noticing that our state-of-the-art speech recognizer has only one scoring pass in a different way from most systems designed for the 1994 ATIS ARPA evaluation (Proceedings of the SLT Workshop—Austin, 1995). The effectiveness (in terms of perplexity performance) and the efficiency (in terms of parameter number) of the VNSAs has greatly contributed to the one-pass system configuration that is also used for our real time system.

8. Conclusions

In this work we have presented an approximation of an n-gram stochastic language model by means of a non-deterministic automaton: the Variable N-gram Stochastic Automaton (VNSA). The VNSA implements the backoff mechanism of the n-gram language model without affecting the performance of the speech decoding algorithm itself (i.e. the backoff is compiled in the network rather than performed at run time). Moreover, the number of parameters of the VNSA is *not* an exponential function of the vocabulary size, but it depends linearly on the number of events (word tuples) in the training set. Furthermore, the number of parameters (transitions and states) can be reduced with a heuristic minimization algorithm with a negligible decrease in the language model performance. As a result, the VNSA networks are a viable approach to efficiently incorporate the n-gram ($n \geqslant 1$) class-based language model in the standard Viterbi one-pass search algorithm for state-of-the-art speech recognition and language understanding.

The authors wish to thank the anonymous reviewers and Allen Gorin whose critical comments have improved the presentation of this paper.

References

Austin, S., Schwartz, R. & Placeway, P. (1991). The forward–backward search algorithm. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Toronto, Canada, pp. 697–700.

Bahl, L. R., Jelinek, F. & Mercer, R. (1983). A maximum likelihood approach to continuous speech recognition. In *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 179–190.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of Markov processes. *Inequalities* **3**, 1–8.

Bell, T., Clearly, J. & Witten, I. H. (1990). *Text Compression.* Prentice Hall, Englewood Cliffs, NJ.

Bocchieri, E., Riccardi, G. & Anantharaman, J. (1994). The AT&T ATIS CHRONUS recognizer. In *Proceedings of the Workshop on Spoken Language Technology*, Austin, TX, U.S.A.

Brown, P., Della Pietra, V. J., de Souza, P. V., Lai, J. C. & Mercer, L. (1990). A statistical approach to machine translation. *Computational Linguistics* **16**, 79–85.

Brown, P., de Souza, P., Mercer, R., Della Pietra, V. & Lai, J. (1992). Class-based n-gram models of natural language. *Computational Linguistics* **18**, 467–479.

Fano, R. M. (1961). *Transmission of Information: a Statistical Theory of Communications.* Wiley, Chichester.

Giachin, E. (1995). Phase bigrams for continuous speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, IL, U.S.A., pp. 225–238.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika* **40**, 237–264.

Jelinek, F. (1976). Continuous speech recognition by statistical methods. In *Proceedings of the IEEE*, pp. 532–556.

Jelinek, F. & Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Pattern Recognition in Practice*, pp. 381–397. North Holland, Amsterdam.

Katz, S. M. (1987). Estimation of probabilities from sparse data from the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pp. 400–401.

Kneser, R. & Ney, H. (1993). Improved clustering techniques for class-based statistical language models. In *Proceedings of EUROSPEECH, European Conference on Speech Communication and Technology*, Berlin, Germany, pp. 973–976.
Magermann, D. & Marcus, M. (1990). Parsing a natural language using mutual information statistics. In *Proceedings of the AAAI*, Boston, MA, U.S.A., pp. 984–989.
Ney, H. & Essen, U. (1993). Estimating small probabilities by leaving-one-out. In *Proceedings of EUROSPEECH, European Conference on Speech Communication and Technology*, Berlin, Germany, pp. 2239–2242.
Pereira, F., Tishby, N. & Lee, L. (1993). Distributional clustering of English words. *30th Annual Meeting of the Association for Cmputational Linguistics*, Columbus, TX, U.S.A., pp. 183–190.
Pereira, F., Riley, M. & Sproat, R. (1994). Weighted rational transductions and their applications to human language processing. In *Proceedings of the Workshop on Human Language Technology*, Austin, TX, U.S.A., pp. 249–254.
Pieraccini, R. & Levin, E. (1992). Stochastic representation of semantic structure for speech understanding. *Speech Communication* **11**, 283–288.
Pieraccini, R. & Levin, E. (1995). CHRONUS, the new generation. In *Proceedings of the Worshop on Spoken Language Technology*, Austin, TX, U.S.A.
Placeway, P., Schwartz, R., Fung, P. & Nguyen, L. (1993). The estimation of powerful language models from small and large corpora. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Minneapolis, MN, U.S.A., pp. 33–36.
Rabiner, L. & Juang, B. H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, NJ.
Riccardi, G., Bocchieri, E. & Pieraccini, R. (1953). Non deterministic stochastic language models for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Detroit, IL, U.S.A., pp. 237–240.
Soong, F. & Huang, E. (1990). A tree-trellis based fast search for finding the *n*-best sentence hypotheses. In *Proceedings of the Workshop on Speech and Natural Language*, Albuquerque, NM, U.S.A.
Stone, M. (1974). Cross validatory choice assessment of statistical predictions. *Royal Statistical Society* **B**, 111–147.
Witten, I. H. & Bell, T. C. (1991). The zero frequency problem: estimating the probabilities of novel events in adaptive text compression. In *IEEE Transactions on Information Theory*, pp. 1085–1093. Englewood Cliffs, NJ, Prentice Hall.

(*Received 1 January 1996 and accepted for publication 11 July 1996*)

Appendix A: discounting techniques

The concept behind discounting techniques can be expressed as follows. Let us assume we are trying to estimate the probabilities of a set of K events, namely $\hat{P}(c_k)$, $k=1, \ldots, K$. From a given training set, each event can be assigned a frequency count $C(c_k)$, $k=1, \ldots, K$. It is useful to define as n_r the number of different events that were observed exactly r times ($r=C(c_k)$) and then we have that:

$$\sum_{r=1}^{R} r n_r = N \tag{A.1}$$

where R is the frequency count of the most numerous events and N is the total number of samples in the corpus. All the events with the same count r have a maximum likelihood estimate, $\hat{P}_{ML}(c_k)$:

$$\hat{P}_{ML}(c_k) = \frac{r}{N} \tag{A.2}$$

Since it is not correct to assign a zero probability to the events that were never observed, a common technique for coping with this problem (called *discounting*) consists

of taking part of the probability mass from the observed events and redistributing it among the unseen and rare events. The most used implicit[4] discounting technique is called the Good–Turing estimation (Good, 1953), and it is defined by the following formula:[5]

$$\hat{P}_{GT}(c_k) = \frac{1}{N}\frac{(r+1)n_{r+1}}{n_r} = \frac{(r+1)n_{r+1}}{rn_r}\hat{P}_{ML}(c_k) = \frac{r^*}{r}\hat{P}_{ML}(c_k) \tag{A.3}$$

In particular, for the unseen events:

$$\hat{P}_{GT}(\textit{unseen events}) = \frac{n_1}{N} \tag{A.4}$$

The problem arising in the implementation of the Good–Turing estimation of the probability of unseen events is that, again due to the limited size of the training corpus, n_r can be 0 for some value of r (e.g. we may not have events that were observed *exactly* five times, but we may have events observed four and six times). Solutions to this problem can be found by smoothing the values n_r (e.g. by using a median filter) or by introducing additional constraints in the probability estimation process, like that suggested by Ney and Essen (1993). Another smoothing procedure for the Good–Turing estimates has been applied by Katz (1987). In this work, the probability mass for the *unseen events* has been collected from the events that have occurred not more than k_s times. As a consequence, the events c_i, such that $C(c_i)>k_s$, are estimated according to the ML formula given in Equation (A.2). Therefore, the probability of event c_i ($C(c_i)>0$) is computed through the following formula:

$$\hat{P}_{KZ}(c_k) = \begin{cases} \hat{P}_{ML}(c_k) & \text{if } C(c_k) \geqslant k_s \\ d_r^{KZ}\hat{P}_{ML}(c_k) & \text{if } 0<C(c_k)<k_s \end{cases} \tag{A.5}$$

where d_r^{KZ} is a discount factor given by ($r = C(c_k)$):

$$d_r^{KZ} = \frac{\frac{r^*}{r} - \frac{(k_s+1)n_{k+1}}{n_1}}{1 - \frac{(k_s+1)n_{k+1}}{n_1}} \quad 1 \leqslant r \leqslant k_s \tag{A.6}$$

where r^* is defined as in Equation (A.3). A more direct approach to the estimation of the discount parameters is used in the *explicit discounting*. With *explicit discounting*, the amount of discounting off the probabilities of observed events is explicitly given as a free parameter. In particular, the following equations hold:

[4] In the Good–Turing estimation the portion of the probability mass that is redistributed among the unseen and rare events is not explicitly defined.

[5] It can be demonstrated (Ney & Essen, 1993) that Equation (A.3) can be derived by applying maximum likelihood estimation of probabilities under the *leave-one-out* paradigm.

$$\hat{P}(c_k)=\frac{(r-d_r)}{N} \tag{A.7}$$

where d_r is given as a function of r and $r=C(c_k)$ as above. The probability mass D available for redistribution is then:

$$D=1-\sum_{r=1}^{R} n_r\hat{P}(c_k)=1-\sum_{r=1}^{R}\frac{n_r(r-d_r)}{N}=\sum_{r=1}^{R}\frac{n_r d_r}{N} \tag{A.8}$$

In the so-called *linear* discounting the discount parameter d_r is directly proportional to r, that is:

$$\begin{aligned} d_r&=\alpha_r r \quad 0\leqslant\alpha\leqslant 1\\ \hat{P}(c_k)&=(1-\alpha_r)\frac{r}{N} \quad r>0 \end{aligned} \tag{A.9}$$

and then:

$$D=\sum_{r=1}^{R}\alpha_r\frac{rn_r}{N} \tag{A.10}$$

A simpler[6] way of approaching the discounting problem is referred to as *uniform linear* discounting. In this case $\alpha_r=\alpha\ \forall r$. Hence, the whole probability mass α is assigned to all unseen events, namely:

$$D=\hat{P}(\textit{unseen events})=\alpha \tag{A.12}$$

In the field of text compression some heuristic techniques have been studied in order to cope with the problem of the unseen events (also addressed as *zero frequency problem*) (Bell, Cleary & Witten, 1990; Witten & Bell, 1991). Below, the methods *add*-1, *sub*-1 and *add*$-c$ are described. Let N be the total number of event observations, c_k the generic event and K the number of distinct events. The method *add*-1 states that the probability of observing a novel event ($\hat{P}$(*unseen events*)) is given by:

$$\hat{P}_{+1}(\textit{unseen events})=\frac{1}{N+1} \tag{A.12}$$

and the probability of a generic event, c_k, with $r=C(c_k)$, is:

$$\hat{P}_{+1}(c_k)=\frac{r}{N+1} \tag{A.13}$$

[6] In this context simplicity is measured by the number of free parameters whose values have to be estimated for the implementation of the language model.

For N large, the method *add*-1 makes $\hat{P}_{+1}(c_k)$ very close to the *ML* estimate $\hat{P}_{ML}(c_k)$ and it assigns a small probability to the unseen events. Besides, the probability of observing a novel event is the same as the one of observing an event c_k with $C(c_k)=1$. The method *sub*-1 makes the assumption that events c_k, such that $C(c_k)=1$, have to be treated as *unseen events.* This way the counts of all events c_k are decreased by one and the collected probability mass is assigned to the novel event:

$$\hat{P}_{-1}(\textit{unseen events})=\frac{K}{N} \tag{A.14}$$

where K is defined as above. The probability of a generic event c_k is:

$$\hat{P}_{-1}(c_k)=\frac{r-1}{N} \tag{A.15}$$

Both $\hat{P}_{+1}(c_k)$ and $\hat{P}_{-1}(c_k)$ estimates are very close when N is large, but the probability space in the case of the method *sub*-1 is differently partitioned. In fact, in the case of method *add*-1, the events c_k, such that $C(c_k)=1$, are not considered *unseen events* as in method *sub*-1. The third method, *add*-*c*, has been thought of as a compromise between the two methods described above. Hence, for method $add-c$, the event space is not modified and events c_k, such that $C(c_k)=1$, still have a small probability. Moreover, the larger the number of distinct events, K, the higher the probability of a *novel event*. In particular, the probability of the *novel event* is:

$$\hat{P}_{+c}(\textit{unseen events})=\frac{K}{N+K} \tag{A.16}$$

and the probability of a generic event c_k is:

$$\hat{P}_{+c}(c_k)=\frac{r}{N+K} \tag{A.17}$$

The underlying reasoning of this estimate is that each time one of the K events has been observed, a *novel event* has also occurred so that the *novel event* has been counted K times.

As a whole, all event estimates presented in this Appendix can be expressed in a general form:

$$\hat{P}(c_k)=a_k\hat{P}_{ML}(c_k)+b_k \tag{A.18}$$

where the coefficients a_k and b_k are derived by inspection of Equations (A.3), (A.7), (A.9), (A.13), (A.15) and (A.17). Within the framework of VNSAs, for each state s with history $v_1, \ldots, v_n$, there is the probability of a *novel event* (backoff probability) and a set of probabilities of events $c_k \equiv \{\textit{word } w_k \textit{ with left context } v_1, \ldots, v_n\}$.

Appendix B: probability estimation with word and selected-phrase classes

The computation of the *n*-gram type probability, by means of word and selected-phrase classes, can be viewed as a smoothing procedure which provides a robust estimate of the word tuples observed in the training data. We examine first the case of word classes and then that of selected-phrases.

In the case of a word class, different words are mapped into the same set so that, for example, "BOSTON" and "CHICAGO" belong to the same class *city.* More precisely, this is a many-to-one mapping $f_C: w \rightarrow \mathscr{C}_j$, from a word w to a class $\mathscr{C}_j$. If we associate each word w in the training set with a class $\mathscr{C}_j$, then we can compute the counts $r_{\mathscr{C}_j}$:

$$r_{\mathscr{C}_j} = C(\mathscr{C}_j) = \sum_{w \in \mathscr{C}_j} C(w) \tag{B.1}$$

This way the probability estimation of class tuples, $(\mathscr{C}_1, \ldots, \mathscr{C}_i)$, benefit from more robust and reliable frequency counts. The probability of the word tuple $(w_1, \ldots, w_{i-1})$ in terms of the class tuple probability can be computed as by Jelinek (1980) and Brown *et al.* (1992):

$$\hat{P}(w_i | w_1, \ldots, w_{i-1}) = \hat{P}(w_i | \mathscr{C}_i) \hat{P}(\mathscr{C}_i | \mathscr{C}_1, \ldots, \mathscr{C}_{i-1}) \tag{B.3}$$

where the generic word w_j is such that $f_C: w_j \rightarrow \mathscr{C}_j$. In Equation (B.2) the *n*-gram probability $\hat{P}(w_i | w_1, \ldots, w_{i-1})$ is factored in two terms: the term $\hat{P}(w_i | \mathscr{C}_i)$ is the probability of word w_i given the class $\mathscr{C}_i$, and $\hat{P}(\mathscr{C}_i | \mathscr{C}_1, \ldots, \mathscr{C}_{i-1})$ can be computed by means of the methods presented in Appendix A.

As for the case of phrase classes, we assume here that the whole set $\mathscr{G}$ of such phrases is given as well as the partition $\mathscr{G}_i$, such that $\mathscr{G} = \cup_i \mathscr{G}_i$ and $\mathscr{G}_i \cap \mathscr{G}_j = \emptyset$. To each subset $\mathscr{G}_i \in \mathscr{G}$ is associated a many-to-one mapping $f_G: g_k = w_i, \ldots, w_j \rightarrow \mathscr{G}_k$. Thus, for each subset $\mathscr{G}_k$ an automaton can be designed in order to recognize all word sequences $g_k = w_i, \ldots, w_j$ mapped into $\mathscr{G}_k$. These automata are used to map a word sequence W into a phrase label tuple $g_1, \ldots, g_M$. If we assume that this is a one-to-one mapping, $\hat{P}(W)$ can be computed as follows:

$$\hat{P}(W) = \prod_{k=1}^{M} \hat{P}(\mathscr{G}_k | \mathscr{G}_1, \ldots, \mathscr{G}_{k-1}) \hat{P}(g_k | \mathscr{G}_k) \tag{B.3}$$

where $f_G: g_k \rightarrow \mathscr{G}_k$. As for the word class case, the estimation is performed in two steps. First, a robust and reliable estimate $\hat{P}(\mathscr{G}_k | \mathscr{G}_1, \ldots, \mathscr{G}_{k-1})$ is obtained and then $\hat{P}(g_k | \mathscr{G}_k)$ gives the probability of subsequence $w_i, \ldots, w_j$, given the phrase set $\mathscr{G}_k$.

As a whole, the same calculation schema for both word class and selected-phrases is a viable approach for the application of these concepts into VNSA automata for language modeling.