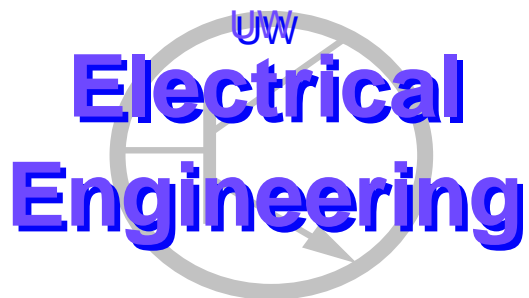

Automatic Learning of Language Model Structure

Kevin Duh, Katrin Kirchhoff
{duh,katrin}@ee.washington.edu

Signal, Speech, and Language Interpretation Lab
Dept of Electrical Engineering, University of Washington
Seattle WA, 98195-2500



UWEE Technical Report
Number UWEETR-2004-0014
April 2004

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: <http://www.ee.washington.edu>

Automatic Learning of Language Model Structure

Kevin Duh, Katrin Kirchhoff
{duh, katrin}@ee.washington.edu

Signal, Speech, and Language Interpretation Lab
Dept of Electrical Engineering, University of Washington
Seattle WA, 98195-2500

University of Washington, Dept. of EE, UWEETR-2004-0014

April 2004

Abstract

In spite of recent advances in statistical algorithms and increased availability of large text corpora, statistical language modeling remains a challenging task, in particular for morphologically rich languages. Recently, new approaches based on factored language models have been developed to address this problem. Whereas standard language models only condition on preceding words, these models provide principled ways of including additional conditioning variables other than the preceding words, such as morphological or syntactic features. However, the number of possible choices for model parameters creates a large space of models that cannot be searched exhaustively. This paper presents an entirely data-driven model selection procedure based on genetic search, which is shown to outperform both knowledge-based and random selection procedures on two different language modeling tasks (Arabic and Turkish).¹

1 Introduction

In spite of novel algorithmic developments and the increased availability of large text corpora, statistical language modeling remains a difficult problem, particularly for languages with rich morphology. Such languages typically exhibit a large number of word types in relation to word tokens in a given text, which leads to high perplexity and a large number of unseen word contexts. As a result, probability estimates are often unreliable, even when using standard smoothing and parameter reduction techniques.

Recently, a new language modeling approach, called *factored language models (FLMs)*, has been developed [34, 4]. FLMs are a generalization of standard language models in that they allow a larger set of conditioning variables for predicting the current word. In addition to the preceding words, any number of additional variables can be included, to represent e.g. morphological, syntactic, or semantic word features. Since such features are typically shared across multiple words, they can be used to obtain better smoothed probability estimates when training data is sparse.

However, the space of possible models is extremely large, due to many different ways of choosing subsets of conditioning word features, backoff procedures, and discounting methods. Usually, this space cannot be searched exhaustively, and optimizing models by a knowledge-inspired manual search procedure usually leads to suboptimal results, since only a small portion of the search space can be explored.

¹This paper is an extended version of the paper that appeared in Coling-2004

In this paper we investigate the possibility of determining the structure of factored language models (i.e. the set of conditioning variables, the backoff procedure and the discounting parameters) by a data-driven search procedure, viz. Genetic Algorithms (GAs). We apply this technique to two different tasks (language modeling for Arabic and Turkish) and show that GAs lead to better models than either knowledge-inspired manual search or random search.

The remainder of this paper is structured as follows: Section 2 describes the details of the factored language modeling approach. The application of GAs to the problem of determining language model structure is explained in Section 3. The corpora used in the present study are described in Section 4 and experiments and results are presented in Section 5. Section 6 compares the present study to related work and Section 7 concludes.

2 Factored Language Models

A standard statistical language model computes the probability of a word sequence $W = w_1, w_2, \dots, w_T$ as a product of conditional probabilities of each word w_i given its history, which is typically approximated by just one or two preceding words (leading to bigrams, and trigrams, respectively). Thus, a trigram language model is described by

$$p(w_1, \dots, w_T) \approx \prod_{i=3}^T p(w_i | w_{i-1}, w_{i-2}) \quad (1)$$

Even with this limitation, the estimation of the required probabilities is challenging: many word contexts may be observed infrequently or not at all, leading to unreliable probability estimates under maximum likelihood estimation [28]. Several techniques have been developed to address this problem, in particular smoothing techniques [8] and class-based language models [5]. Smoothing methods include discounting the maximum likelihood estimates [16, 53], linearly interpolating with lower-order n-grams [29], and recursively backing off to lower-order n-grams [32]. Class-based models alleviate the problem of data sparsity by clustering the vocabulary. Let C_i be the class that word w_i is assigned to, then the joint probability can be described by a hidden Markov model (HMM) with words as observations and classes as hidden states. Assuming a deterministic mapping of w_i to C_i , a bigram would be:

$$Pr(w_2 | w_1) = Pr(w_2 | C_2) Pr(C_2 | C_1) \quad (2)$$

The word classes can be derived either using linguistic knowledge or data-driven clustering techniques [5, 42].

In spite of such parameter reduction techniques, language modeling remains a difficult task, in particular for morphologically rich languages, e.g. Turkish, Russian, or Arabic. Such languages have a large number of word types in relation to the number of word tokens in a given text, as has been demonstrated in a number of previous studies [17, 33, 23, 43]. This in turn results in a high perplexity and in a large number of out-of-vocabulary (OOV) words when applying a trained language model to a new unseen text.

An illustration of the morphologically-rich nature of Arabic is shown in Figures 2 and 2. Figure 2 plots the vocabulary growth rates of the English and Arabic CallHome corpora, displaying the number of unique words as the corpus size (i.e. number of word tokens) is increased. From Figure 2, we observed that the vocabulary growth rate of Arabic words is substantially higher than that of English. Figure 2 shows the vocabulary growth rate from the stemmed version of the same corpora. The Arabic text was stemmed by looking up the stem for each word in the CallHome lexicon; the English text was stemmed by the Porter stemmer [48]. As expected, we observed a reduction in vocabulary growth rate in both languages relative to Figure 2. However, the reduction in Arabic is much greater than the reduction in English. This demonstrates

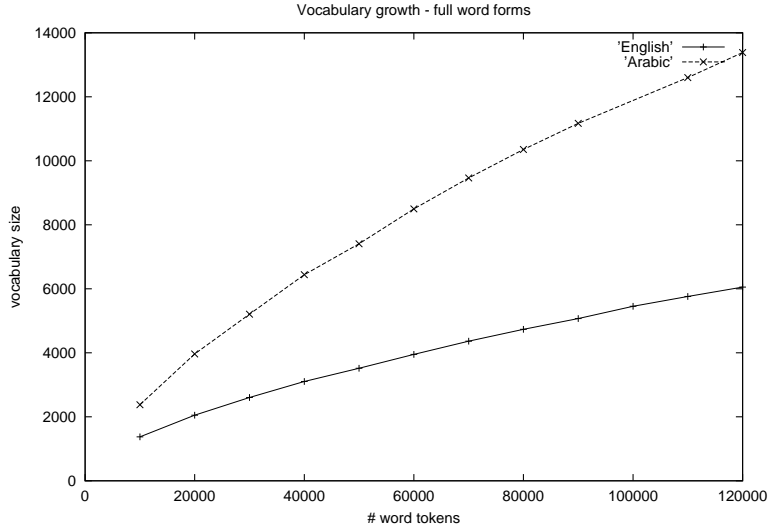


Figure 1: Vocabulary growth rate of English and Arabic, full word forms.

that most of the vocabulary growth in Arabic is indeed caused by the multiplication of word forms due to morphological affixation.

Given the difficulties in language modeling based on full word forms it would be desirable to find a way of decomposing word forms into their morphological components and to build a more robust language model based on probabilities involving individual morphological components.

2.1 Factored Word Representations

A recently developed approach that addresses this problem is that of *Factored Language Models (FLMs)* [34, 4], whose basic idea is to decompose words into sets of features (or *factors*) instead of viewing them as unanalyzable wholes. Probabilistic language models can then be constructed over (sub)sets of word features instead of, or in addition to, the word variables themselves. For instance, words can be decomposed into stems/lexemes and POS tags indicating their morphological features, as shown below:

<i>Word:</i>	Stock	prices	are	rising
<i>Stem:</i>	Stock	price	be	rise
<i>Tag:</i>	Nsg	N3pl	V3pl	Vpart

Such a representation serves to express lexical and syntactic generalizations, which would otherwise remain obscured. It is comparable to class-based representations employed in class-based models; however, in FLMs several simultaneous class assignments are allowed instead of a single one. In general, we assume that a word is equivalent to a fixed number (K) of factors, i.e. $W \equiv f^{1:K}$. The task then is to produce a statistical model over the resulting representation - using a trigram approximation, the resulting probability model is as follows:

$$p(f_1^{1:K}, f_2^{1:K}, \dots, f_T^{1:K}) \approx \prod_{t=3}^T p(f_t^{1:K} | f_{t-1}^{1:K}, f_{t-2}^{1:K}) \quad (3)$$

Thus, each word is dependent not only on a single stream of temporally ordered word variables, but also on additional parallel (i.e. simultaneously occurring) features. This factored representation can be used in two different ways to improve over standard LMs, either by using a product model or a backoff model.

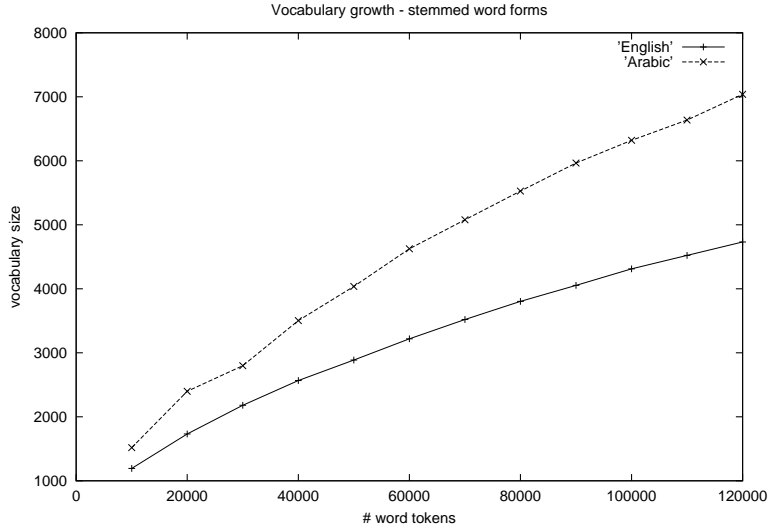


Figure 2: Vocabulary growth rate of English and Arabic, stemmed words.

The product model uses conditional independence statements to write the word probabilities in factored form. Consider a factored representation consisting of the word W , its stemmed version S , and its part-of-speech (POS) tag C . When making the following assumptions

1. the current word is independent of all previous factors given the current stem and POS tag
2. the current stem, given the previous stem, is independent of all other previous factors, and
3. the current POS tag, given the previous POS tag, is also independent of all other previous factors

the product model (in terms of a bigram) will have the following form:

$$p(w_t|s_t, c_t)p(s_t|s_{t-1})p(c_t|c_{t-1}) \quad (4)$$

We can expect low perplexity for the first term $p(w_t|s_t, c_t)$ since knowledge of both the stem and POS tag gives much information about the word itself. Therefore, if both $p(s_t|s_{t-1})$ and $p(c_t|c_{t-1})$ have low enough perplexity, then it is possible for the product model to achieve an overall perplexity that is lower than that of a standard word bigram.

In contrast, the backoff model approach obtains word probabilities by only conditioning on factors at previous time-steps and backing off to subsets of these factors in case of insufficient data. For instance, using the factored representation above, a bigram model could be formulated as:

$$p(w_t|w_{t-1}, s_{t-1}, c_{t-1}) \quad (5)$$

If the counts $(w_t, w_{t-1}, s_{t-1}, c_{t-1})$ is insufficient during training, a lower order model such as $p(w_t|s_{t-1}, c_{t-1})$ can be used to obtain probabilities. It should be noted that adding s_{t-1} and c_{t-1} to the standard word bigram model $p(w_t|w_{t-1})$ to get $p(w_t|w_{t-1}, s_{t-1}, c_{t-1})$ gives no information gain since the stems and POS tags are often deterministic functions of the word. Nevertheless, this method achieves more robust estimation in cases where the standard word bigrams occur infrequently. For instance, if a word bigram does not occur in training data, but the stem and POS tags do occur, then backoff achieves more robust estimation. This is often the case for morphologically-rich languages. Therefore, we focus on backoff models in this paper.

Finally, it should be noted that the factored representation is quite general. Although we use only linguistically-motivated factors in this work, any knowledge-engineered or data-driven word classes can be used as factors.

2.2 Generalized Parallel Backoff

A particular issue that arises with the backoff model is the order of backoff. In standard word n-grams, backoff usually proceeds by dropping the most distant word first. The assumption is that distant words have less predictive power than close words. In the factored representation, many factors may occur parallel in time, so it is not immediately obvious which factors should be dropped first. For example, to back off from the model $p(w_t|s_{t-1}, c_{t-1})$, either s_{t-1} or c_{t-1} could be dropped, but we do not know a priori which one leads to more robust estimates. This section introduces *Generalized Parallel Backoff*, which solves this problem in a flexible manner.

Backoff [32] is a common smoothing technique in language modeling. It is applied whenever the count for a given n-gram in the training data falls below a certain threshold τ . In that case, the maximum-likelihood estimate of the n-gram probability is replaced with a probability derived from the probability of the lower-order $(n - 1)$ -gram and a backoff weight. N-grams whose counts are above the threshold retain their maximum-likelihood estimates, discounted by a factor that re-distributes probability mass to the lower-order distribution:

$$p_{BO}(w_t|w_{t-1}, w_{t-2}) = \begin{cases} d_c p_{ML}(w_t|w_{t-1}, w_{t-2}) & \text{if } c > \tau_3 \\ \alpha(w_{t-1}, w_{t-2}) p_{BO}(w_t|w_{t-1}) & \text{otherwise} \end{cases} \quad (6)$$

Here c is the count of (w_t, w_{t-1}, w_{t-2}) in the training data, probability p_{ML} denotes the maximum-likelihood estimate and d_c is a discounting factor that is applied to the higher-order distribution. The way in which the discounting factor is estimated determines the actual *smoothing* method (e.g. Good-Turing, Kneser-Ney, etc.) The normalization factor $\alpha(w_{t-1}, w_{t-2})$ ensures that the entire distribution sums to one. (See [8] for an overview.) During standard backoff, the most distant conditioning variable (in this case w_{t-2}) is dropped first, then the second most distant variable etc. until the unigram is reached. This can be visualized as a backoff *path* (Figure 3(a)). If the only variables in the model are words, such a backoff procedure is re-

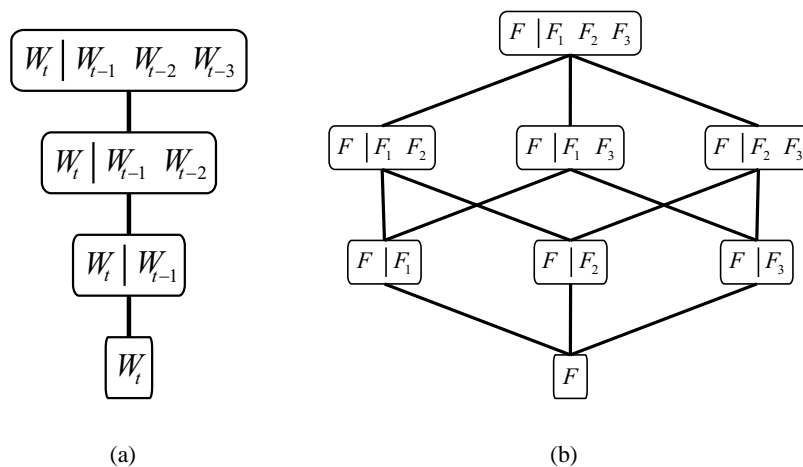


Figure 3: Standard backoff path for a 4-gram language model over words (left) and backoff graph for 4-gram over factors (right).

sonable. However, if variables occur in parallel, i.e. do not form a temporal sequence, it is not immediately obvious in which order they should be dropped. In this case, several backoff paths are possible, which can be summarized in a backoff *graph* (Figure 3(b)). In the example, the node $p(F|F_1, F_2, F_3)$ has three options, those corresponding to dropping any one of the three parents. The nodes $p(F|F_1, F_2)$, $p(F|F_1, F_3)$, and $p(F|F_2, F_3)$ each have two options, and the nodes $p(F|F_1)$, $p(F|F_2)$, and $p(F|F_3)$ each have only one option.

In principle, there are several different ways of choosing among different paths in this graph:

1. Choose a fixed, predetermined backoff path based on linguistic knowledge, e.g. always drop syntactic before morphological variables.
2. Choose the path at run-time based on statistical criteria (e.g. counts of n-gram in training data).
3. Choose multiple paths and combine their probability estimates.

The last option, referred to as *parallel* backoff, is implemented via a new, generalized backoff function (here shown for a 4-gram):

$$p_{GBO}(f|f_1, f_2, f_3) = \begin{cases} d_c p_{ML}(f|f_1, f_2, f_3) & \text{if } c > \tau_4 \\ \alpha(f_1, f_2, f_3)g(f, f_1, f_2, f_3) & \text{otherwise} \end{cases} \quad (7)$$

where c is the count of (f, f_1, f_2, f_3) , $p_{ML}(f|f_1, f_2, f_3)$ is the maximum likelihood distribution, τ_4 is the count threshold, and $\alpha(f_1, f_2, f_3)$ is the normalization factor. The function $g(f, f_1, f_2, f_3)$ determines the backoff strategy. In a typical backoff procedure $g(f, f_1, f_2, f_3)$ equals $p_{BO}(f|f_1, f_2)$. In generalized parallel backoff, however, g can be *any non-negative function* of f, f_1, f_2, f_3 . In our implementation of FLMs [43] we consider several different g functions, including the mean, weighted mean, product, and maximum of the smoothed probability distributions over all subsets of the conditioning factors. In addition to different choices for g , different discounting parameters can be chosen at different levels in the backoff graph. For instance, at the topmost node, Kneser-Ney discounting might be chosen whereas at a lower node Good-Turing might be applied.

FLMs have been implemented as an add-on to the widely-used SRILM toolkit [50, 4] and have been used successfully for the purpose of morpheme-based language modeling [4], multi-speaker language modeling [30], language identification [47] and speech recognition [43].

3 Learning FLM Structure

In order to use an FLM, three types of parameters need to be specified: the initial conditioning factors, the backoff graph, and the smoothing options. The initial conditioning factors specifies which factors shall be used in estimating n-gram probabilities. The backoff graph and smoothing options indicate the procedures for robust estimation in the case of insufficient data. The goal of structure learning is to find the parameter combinations that create FLMs that achieve a low perplexity on unseen test data.

The resulting model space is extremely large: given a factored word representation with a total of k factors, there are $\sum_{n=1}^k \binom{k}{n}$ possible subsets of initial conditioning factors. For a set of m conditioning factors, there are up to $m!$ backoff paths, each with its own smoothing options. Unless m is very small, exhaustive search is infeasible.

Moreover, nonlinear interactions between parameters make it difficult to guide the search into a particular direction. For instance, a particular backoff path may work well with Kneser-Ney smoothing, while a slightly different path may find poor performance using the same method. Good structures are ultimately dependent on the data, so parameter sets that work well for one corpus cannot necessarily be expected to perform well on another.

In the following section, we describe an automatic structure learning technique, based on Genetic Algorithms, that performs well despite the search space challenges.

3.1 Genetic Algorithms

Genetic Algorithms (GAs) [27] are a class of evolution-inspired search/optimization techniques. They perform particularly well in problems with complex, poorly understood search spaces.

The fundamental idea of GAs is to encode problem solutions as strings (genes), and to evolve successive populations of solutions through the use of genetic operators. Solutions are evaluated according to a fitness function which represents the desired optimization criterion.

The individual steps in the algorithm are as follows:

Initialize: Randomly generate a set (population) of strings (genes).

While average fitness of population improves by a certain threshold:

Evaluate fitness: calculate each string's fitness

Apply operators: apply the following genetic operators to create a new population.

-*Selection* Probabilistically select genes to populate the next generation based on the fitness value.

-*Crossover* Exchange substrings of two genes to generate new solutions.

-*Mutation* With some probability, randomly alter a substring in the gene to create population diversity.

Figure 4 illustrates the application of genetic operators to a population of four binary strings. First, the fitness of each string is evaluated using the fitness function. Then, strings are selected to the new gene pool by probabilistic selection. Finally, crossover and mutation alter the selected strings to create the population for the next generation. This process iterates until convergence.

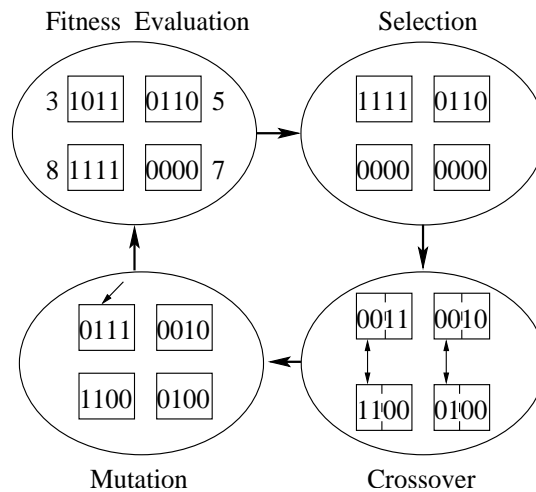


Figure 4: Illustration of Genetic Operators in GA search. The population consists of four genes, each a four-bit binary string. Populations of solutions evolve by repeatedly evaluating fitness and applying selection, crossover, and mutation to the genes.

Although GAs provide no guarantee of finding the optimal solution, they often find good solutions quickly. Their success is attributed to several characteristics: First, by maintaining a population of solutions rather than a single solution, GA search is robust against premature convergence to local optima. Second, by optimizing solutions based on a *task-specific* fitness function, and applying genetic operators probabilistically, GA helps direct search towards promising regions of the search space. In particular, the

crossover operator distinguishes GA search from other stochastic search techniques. For a more theoretical explanation of why GAs work, refer to [20, 45].

GAs have been applied to a wide variety of fields, but they can be roughly divided into two broad categories [45]. One category utilizes GA's search/optimization ability as problem solvers; these include applications in automatic programming [10], learning classifier systems [26], VLSI layout [9], time-series prediction [44], molecular biology [49], job scheduling [13], and ergonomic keyboard design [12]. The second category uses GAs to model and simulate natural systems; applications include models for ecosystems [14], for effects of learning on evolution [25], for competitive games, [1], and for immune systems [15]. In the field of speech and language processing, GAs have been used in text classification [39], hidden Markov model (HMM) training [38], phonotactics [3], and grammar induction [40]. [36] contains a survey of GA applications in natural language processing. GA's widespread applicability is due to the fact that they can be tailored to any problem as long as a fitness function and encoding method are specified. In the following section, we describe how we tailor GAs to our specific problem of FLM structure search.

3.2 Structure Search Using GA

To apply GA search to the problem of optimizing FLM parameters, we define a fitness function and an encoding/decoding scheme that maps between a FLM and a string (gene). The fitness value of each gene is defined as the inverse of its perplexity, so that genes with low perplexity will receive a higher probability of selection. In a GA run, most computation will be spent on training FLMs and evaluating their corresponding perplexity. Figure 5 illustrates the high-level operation of a GA search for FLM structures. The main chal-

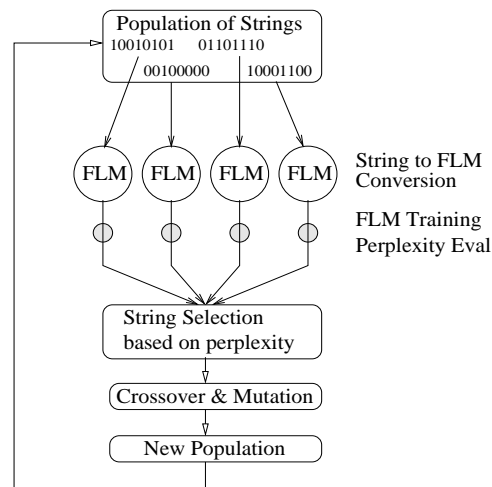


Figure 5: Overall program flow of GA search for FLM structure.

lenges in this operation is the design of the encoding method, or the mapping of FLM parameters to strings. Since genetic operators are directly applied to the encoded strings, an appropriate representation of FLM parameters that is amenable to the application of genetic operators is important. Ideally, substrings should consist of meaningful subparts of a FLM so that the crossover operation can combine good sub-solutions. Moreover, crossover and mutation operators should create new strings that are valid FLM structures. Filters can be used to disallow invalid strings, yet this would become inefficient if invalid strings are produced too often.

Various encoding methods exist, and appropriate encoding methods are often problem-specific. A fixed-length binary string encoding is one of the most common encoding schemes due to its simplicity. In this encoding, solution parameters can either be represented by individual bits or multiple bits. In cases where

multiple-bit encodings are unwieldy (e.g. leading to long strings), k-ary character or real-value encodings may be used instead. More complex problems may require alternative methods that allow representations that are more expressive than fixed-length strings. For instance, tree encoding [37] encodes the gene as a tree structure and allows genes to grow in an open-ended manner (i.e. new subtrees may be attached to the original tree). Tree encoding is now its own field called Genetic Programming. Diploid encodings [24] uses two parallel strings as genes, thereby achieving the effects of dominance and recessive gene behavior in genetics. Messy GAs [19] allow genes of different lengths to be over-specified or under-specified. It is motivated by the biological idea that a complex organism's lengthy genes evolved gradually from shorter genes of more simple organisms. Therefore, Messy GAs allow variable gene lengths in the population and provide mappings to solutions even when the gene is too short (under-specified) or too long (over-specified). Further, it allows for combinations of shorter genes to form longer genes.

Due to the heterogeneous nature of FLM parameters, we design a separate encoding method for each type of parameter. The gene is therefore the concatenation of the three substrings representing conditioning factors, backoff graph, and smoothing options. We explain each encoding scheme in detail in the following subsections.

3.2.1 Conditioning Factors Encoding

The initial factors F are encoded as binary strings, with 0 meaning absence and 1 meaning presence of a factor. For example, suppose a FLM trigram has 3 factors (A,B,C) per word. Then the full factor set S is:

$$\{A_{-1}, B_{-1}, C_{-1}, A_{-2}, B_{-2}, C_{-2}\}$$

where the subscript indicates the time position of each factor. The set of initial factors F is a (sub)set of S and can be represented as a 6 bit binary string, where 1 at a position indicates the inclusion of that factor in F . For instance, the string 10011 would mean that F is:

$$\{A_{-1}, B_{-2}, C_{-2}\}$$

This implies a trigram FLM that is described by:

$$p(f_1^{1:3}, f_2^{1:3}, \dots, f_T^{1:3}) \approx \prod_{t=3}^T p(f_t^{1:3} | A_{t-1}, B_{t-1}, C_{t-1}, A_{t-2}, B_{t-2}, C_{t-2}) \quad (8)$$

$$\approx \prod_{t=3}^T p(f_t^{1:3} | A_{t-1}, B_{t-2}, C_{t-2}) \quad (9)$$

In general, a string has length K for K potential conditioning factors. By manipulating the bits in this string, genetic operators create different conditioning factor subsets F_1, F_2, \dots, F_n , where $n = \sum_{i=1}^K \binom{K}{i}$ is the number of possible combinations. In theory, the ordering of factors in S affects performance since under crossover operation, factors that are close together in position remain together with higher probability than factors that are far apart. In practice, the ordering of factors in S can easily be permuted for various GA runs. Nevertheless, experiments with FLM structure search has shown little difference for different orderings of conditioning factors.

3.2.2 Backoff Graph Encoding

The encoding of the backoff graph is more difficult because of the large number of possible paths. A direct approach encoding every edge as a bit would result in overly long strings, rendering the search inefficient.

There are $m!$ possible backoff paths for an initial set of m conditioning factors, so direct encoding is not scalable for anything but the smallest numbers of factors.

Our solution is to encode a backoff graph in terms of graph grammar rules and use these compact rules to generate the backoff graph. This method is similar in approach to [35, 21, 2], which used different types of grammars to represent and evolve neural networks.

Graph grammar rules are used to encode the graph regularity that a node with m factors can only back off to children nodes with $m - 1$ factors. For instance, for $m = 3$, the choices for proceeding to the next-lower level in the backoff graph can be described by only three grammar rules:

$$\text{RULE 1: } \{x_1, x_2, x_3\} \rightarrow \{x_1, x_2\}$$

$$\text{RULE 2: } \{x_1, x_2, x_3\} \rightarrow \{x_1, x_3\}$$

$$\text{RULE 3: } \{x_1, x_2, x_3\} \rightarrow \{x_2, x_3\}$$

Here x_i corresponds to the factor at the i th position in the parent node. Rule 1 indicates a backoff that drops the third factor, Rule 2 drops the second factor, and Rule 3 drops the first factor.

To describe the backoff from $\{A_{-1}, B_{-2}, C_{-2}\}$ to $\{A_{-1}, C_{-2}\}$, for instance, we would indicate that Rule 2 was activated (the second node was dropped). This implies the backoff function:

$$p_{GBO}(f|f_{-1}^1, f_{-2}^2, f_{-2}^3) = \begin{cases} d_c p_{ML}(f|f_{-1}^1, f_{-2}^2, f_{-2}^3) & \text{if } c > \tau_3 \\ \alpha(f_{-1}^1, f_{-2}^3)g(f, f_{-1}^1, f_{-2}^3) & \text{otherwise} \end{cases} \quad (10)$$

In addition, to describe a parallel backoff from $\{A_{-1}, B_{-2}, C_{-2}\}$ to both $\{A_{-1}, C_{-2}\}$ and $\{B_{-2}, C_{-2}\}$ we would indicate that *both* Rule 2 and Rule 3 are activated. In this case, the backoff function is similar to equation 10, but the function $g()$ would be some combination of $g(f, f_{-1}^1, f_{-2}^3)$ and $g(f, f_{-2}^2, f_{-2}^3)$.

In the case where no rules are specified, “skipping” occurs. Skipping is the simultaneous dropping of two conditioning variables and may be desired when both variables do not give robust estimates. It is so named because it corresponds to skipping an entire level of nodes in the backoff graph. For instance, a backoff from $\{A_{-1}, B_{-2}, C_{-2}\}$ to $\{A_{-1}\}$ is an example of skipping, where both B_{-2} and C_{-2} are dropped simultaneously if there are insufficient counts for (A_{-1}, B_{-2}, C_{-2}) .

The choice of rules used to generate the backoff graph is encoded in a binary string, with 1 indicating the use and 0 indicating the non-use of a rule. The backoff graph grows according to the rules specified by the gene, as shown schematically in Figure 6.

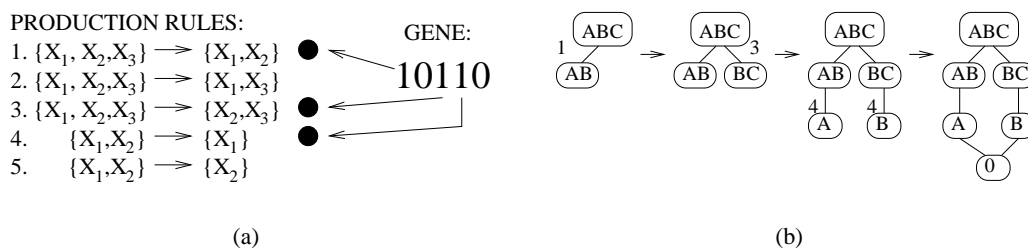


Figure 6: Gene activates graph grammar production rules (a); Generation of Backoff graph by activated rules 1, 3, 4 (b).

In this example, we generate a backoff graph from initial conditioning factors $F = \{A_{-1}, B_{-2}, C_{-2}\}$ using the information represented in the gene 11001. First, all rules are enumerated and the gene specifies which rules are activated. The bits 1 in the first, second, and last position of the gene indicates that Rules 1, 2, and 5 are activated. Then, for all nodes that apply, an activated grammar rule produces backoff children.

In the first production, both Rules 1 and 3 are applied to create two children (parallel backoff). In the second production, Rule 5 can be applied to both AB and BC, thereby producing 1-factor nodes A and B. This is done recursively until we reach one factor, which automatically backs off to the unigram.

If no rules apply, skipping occurs. For instance, if the gene were 00010, none of the rules with 3 factors on the left-hand-side are activated. Therefore, two factors are dropped simultaneously and we proceed to decode using the next lower-level rules. In practice, this algorithm works by applying all 3-factor rules but also noting that the generated nodes would not be considered in the backoff sequence. This ensures the applicability of the next lower-level rules so that graph generation will continue until the unigram node.

In total, this encoding scheme requires a gene of length $\sum_{i=1}^m i$ rather than $m!$. This allows us to encode a graph using few bits but does not represent all possible graphs. We cannot selectively apply different rules to different nodes at the same level – this would essentially require a context-sensitive grammar, which would in turn increase the length of the encoded strings. Figure 7 illustrates the kind of graph structures that cannot be represented. We therefore have a fundamental tradeoff between the most general representation and an encoding that is tractable. Our experimental results described below confirm, however, that sufficiently good results can be obtained in spite of the above limitation.

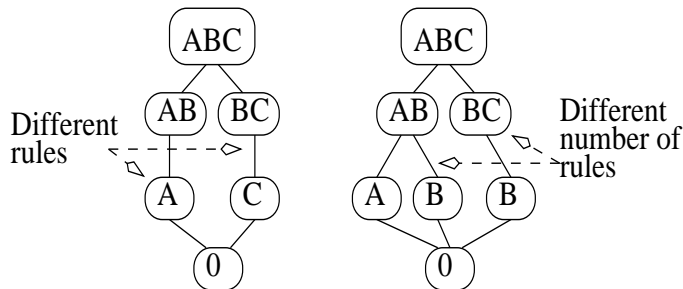


Figure 7: Two examples of backoff graphs that cannot be represented with a context-free grammar since rules cannot be selectively applied to nodes at the same level.

3.2.3 Smoothing Options Encoding

Smoothing options are encoded as tuples of integers. The first integer specifies the discounting method (e.g. Kneser-Ney, Witten-Bell, absolute discounting) and the second integer specifies the minimum count for the n-gram to be included in the language model. The integer string consists of successive concatenated tuples, each representing the smoothing option at a node in the graph. More smoothing options, such as parallel backoff options, may be encoded. However, this will increase the length of the gene, so care must be taken.

The GA operators are applied to concatenations of all three substrings describing the set of factors, backoff graph, and smoothing options, such that all parameters are optimized jointly. In the following sections, we explain the data and experimental results for our GA search algorithm.

4 Data

We tested our language modeling algorithms on two different data sets from two different languages, Arabic and Turkish.

The Arabic data set was drawn from the CallHome Egyptian Conversational Arabic (ECA) corpus [41]. The training, development, and evaluation sets contain approximately 170K, 32K, and 18K words, respec-

tively. The corpus was collected for the purpose of speech recognizer development for conversational Arabic, which is mostly dialectal and does not have a written standard. No additional text material beyond transcriptions is available in this case; it is therefore important to use language models that perform well in sparse data conditions.

The factored representation was constructed using linguistic information from the corpus lexicon, in combination with automatic morphological analysis tools. It includes, in addition to the word, the stem, a morphological tag, the root, and the pattern. The latter two are components which when combined form the stem. An example of this factored word representation is shown below:

Word: il+dOr / *Morph:* noun+masc-sg+article / *Stem:* dOr / *Root:* dwr / *Pattern:* CCC

For our Turkish experiments we used a morphologically annotated corpus of Turkish [22]. The annotation was performed by applying a morphological analyzer, followed by automatic morphological disambiguation as described in [23]. The morphological tags consist of the initial root, followed by a sequence of inflectional groups delimited by derivation boundaries (^DB). A sample annotation (for the word *yararlanmak*, consisting of the root *yarar* plus three inflectional groups) is shown below:

yararmanlak:

yarar+Noun+A3sg+Pnon+Nom / ^DB+Verb+Acquire+Pos / ^DB+Noun+Inf+A3sg+Pnon+Nom

We removed segmentation marks (for titles and paragraph boundaries) from the corpus but included punctuation. Words may have different numbers of inflectional groups, but the FLM representation requires the same number of factors for each word; we therefore had to map the original morphological tags to a fixed-length factored representation. This was done using linguistic knowledge: according to [46], the final inflectional group in each dependent word has a special status since it determines inflectional markings on head words following the dependent word. The final inflectional group was therefore analyzed into separate factors indicating the number (N), case (C), part-of-speech (P) and all other information (O).

Additional factors for the word are the root (R) and all remaining information in the original tag not subsumed by the other factors (G). The word itself is used as another factor (W). Thus, the above example would be factorized as follows:

W: yararlanmak / R: yarar / P: NounInf-N:A3sg / C: Nom / O: Pnon / G:
NounA3sgPnonNom+Verb+Acquire+Pos

Other factorizations are certainly possible; however, our primary goal is not to find the best possible encoding for our data but to demonstrate the effectiveness of the FLM approach, which is largely independent of the choice of factors. For our experiments we used subsets of 400K words for training, 102K words for development and 90K words for evaluation.

5 Experiments and Results

In our application of GAs to language model structure search, the perplexity of models with respect to the development data was used as an optimization criterion. The perplexity of the best models found by the GA were compared to the best models identified by a lengthy manual search procedure using linguistic knowledge about dependencies between the word factors involved, and to a random search procedure which evaluated the same number of strings as the GA.

Table 1 shows the various GA options we tried. (See [20, 45] for explanations of the various GA options.) The following options gave good results: population size 30-50, crossover probability 0.9, mutation population 0.01, Stochastic Universal Sampling as the selection operator, 2-point crossover. We also experimented with re-initializing the GA search with the best model found in previous runs. This is done by

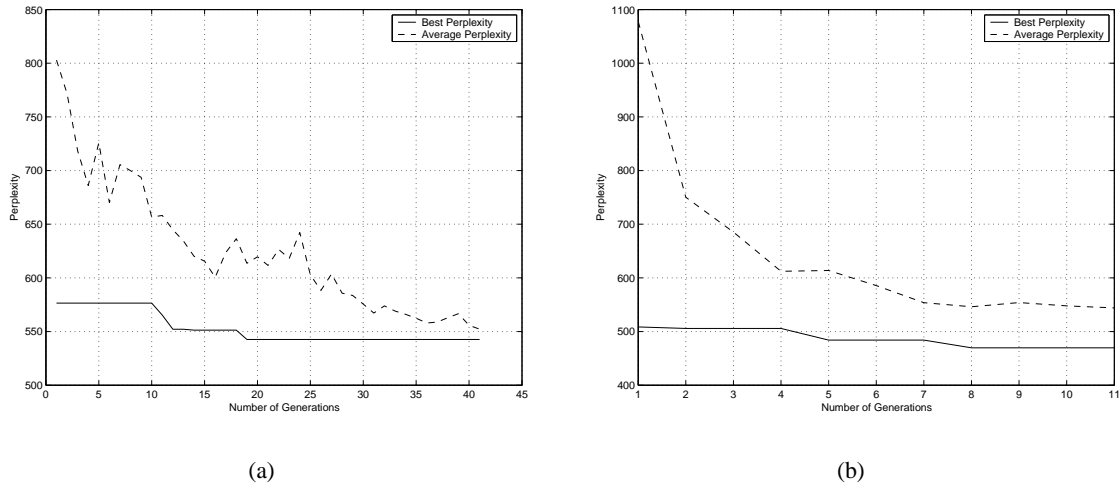


Figure 8: Illustration of fitness optimization in two sample GA runs. (a) is a conventional GA search for Turkish bigrams. (b) uses an elitist GA search for Turkish trigrams. The elitist strategy guarantees that the best-fit gene remains in the population.

“seeding” one or two of the current best models in the initial population. This method consistently improved the performance of normal GA search and we used it as the basis for the results reported below.

Figures 8(a) and 8(b) show two example plots of GA runs, which illustrate the decrease in average and best perplexity as the generation increase. The best perplexity is taken as the best single perplexity value up to the current generation. The average perplexity is the mean of all perplexity values in the population. When the two perplexity values converge, the GA search is terminated. Note that the average perplexity curve is non-monotonic in Figure 8(a) but monotonic in Figure 8(b). The difference is that the GA run in Figure 8(b) uses an additional “elitist selection strategy” [31]. In elitist selection, the best-fit gene of each generation is guaranteed to be selected for the next generation. Such strategy improves the speed to convergence but increases the risk of saturation at local optima. In our experiments, both non-elitist and elitist strategies have been tried, but little difference have been observed. The results below are based on elitist strategies.

Types of GA Options	
Population Size	10-200 strings per generation
Max Generation	10-100
Crossover Probability	.70 - .99
Mutation Probability	.01 - .10
Selection Operator	Roulette Wheel, Stochastic Universal Sampling, Tournament Selection
Crossover Operator	1-point, 2-point, uniform

Table 1: Several GA Options tried in experiments.

5.1 Perplexity Results for Turkish

N	Word	Hand	Rand	GA	GAsced	Δ (%)
Dev Set						
2	593.8	555.0	556.4	542.6	539.2	-2.9
3	534.9	533.5	497.1	469.7	444.5	-10.6
4	534.8	549.7	566.5	551.2	522.2	-5.0
Eval Set						
2	609.8	558.7	525.5	493.4	487.8	-7.2
3	545.4	583.5	509.8	478.5	452.7	-11.2
4	543.9	559.8	574.6	560.5	527.6	-5.8

Table 2: Perplexity for Turkish language models. N = n-gram order, Word = word-based models, Hand = manual search, Rand = random search, GA = genetic search, GAsced = GA with seeding. Δ = relative improvement of GAsced over the best of manual and random search.

Due to the large number of factors in the Turkish word representation, models were only optimized for conditioning variables and backoff paths, but not for smoothing options. Table 2 compares the best perplexity results for standard word-based models and for FLMs obtained using manual search (Hand), random search (Rand), GA search (GA), and GA search with seeding (GAsced). The last column shows the relative change in perplexity for the GA compared to the better of the manual or random search models. For tests on both the development set and evaluation set, GA search gave the lowest perplexity.

5.2 Perplexity Results for Arabic

In the case of Arabic, the GA search was performed over conditioning factors, the backoff graph, and smoothing options. The results in Table 3 were obtained by training and testing without consideration of out-of-vocabulary (OOV) words. Our ultimate goal is to use these language models in a speech recognizer with a fixed vocabulary, which cannot recognize OOV words but requires a low perplexity for other word combinations. In a second experiment, we trained the same FLMs from Table 3 with OOV words included as the unknown word token. Table 4 shows the results. Again, we see that the GA outperforms other search methods. This implies that GA optimization does not overfit and generalizes quite well.

The best language models all used parallel backoff and different smoothing options at different backoff graph nodes. The Arabic models made use of all conditioning variables (Word, Stem, Root, Pattern, and Morph) whereas the Turkish models used only the W, P, C, and R variables (see above Section 4). Refer to the Appendix for examples of the discovered language models in the SRILM FLM file format.

6 Related Work

Various previous studies have investigated the feasibility of using units other than words for language modeling (e.g. [17, 7, 33, 52, 6]). However, in all of these studies words were decomposed into linear sequences of morphs or morph-like units, using either linguistic knowledge or data-driven techniques. Standard language models were then trained on the decomposed representations. The resulting models essentially express statistical relationships between morphs, such as stems and affixes. For this reason, a context larger than that provided by a trigram is typically required, which quickly leads to data-sparsity. In contrast to these approaches, factored language models encode morphological knowledge not by altering the linear segmentation of words but by encoding words as parallel bundles of features.

N	Word	Hand	Rand	GA	GAseed	Δ (%)
Dev Set						
2	229.9	229.6	229.9	223.6	222.9	-2.9
3	229.3	226.1	230.3	217.3	212.6	-6.0
Eval Set						
2	249.9	230.1	239.2	227.2	223.6	-2.8
3	285.4	217.1	224.3	210.5	206.2	-5.0

Table 3: Perplexity for Arabic language models (w/o unknown words).

N	Word	Hand	Rand	GA	GAseed	Δ (%)
Dev Set						
2	236.0	195.5	198.5	193.9	193.3	-1.1
3	237.0	199.0	202.0	192.3	188.1	-5.5
Eval Set						
2	235.2	234.1	247.7	230.9	233.4	-0.7
3	253.9	229.2	219.0	217.4	212.2	-3.1

Table 4: Perplexity for Arabic language models (with unknown words).

The general possibility of using multiple conditioning variables (including variables other than words) has also been investigated by [11, 18, 51, 54]. In [11] and [54], the additional conditioning variables are word classes derived by data-driven clustering procedures, which are then arranged in a backoff lattice or graph similar to the our procedure. Gildea [18] determines the conditional probability of semantic roles given multiple heterogeneous features of syntactic constituents, e.g. position, voice, etc. All of these studies assume a fixed path through the graph, which is usually obtained by an ordering from more specific probability distributions to more general distributions. Some schemes also allow two or more paths to be combined by weighted interpolation. FLMs, by contrast, allow different paths to be chosen at run-time. They support a wider range of combination methods for probability estimates from different paths, and they offer a choice of different discounting options at every node in the backoff graph. Most importantly, none of these studies propose a data-driven method for optimizing the backoff path or initial set of conditioning variables.

The application of graph grammar to gene encodings was first attempted in the context of evolving neural networks [35, 21, 2]. Researchers in this community were interested in simulating the relationship between learning (as modeled by the neural network) and evolution (as modeled by the GA). Grammar encoding was a solution to the problem of inscalability when directly encoding large neural network as genes. [35] observed that regular structures exist within neural networks and therefore defined grammar rules for them. [21] further extended the approach by incorporating ideas from cellular automata theory. Our approach for grammar encoding of backoff graphs arose from the same needs as these work, but differ significantly in terms of the actual encoding scheme. This is simply due to the difference between neural network and backoff graph structures. For instance, whereas [35] explicitly encodes both grammar rules and terminal nodes in the gene, our approach uses grammar rules to generate backoff nodes.

Most importantly, however, the present study is to our knowledge the first to describe an entirely data-driven procedure for identifying the best combination of parameter choices for language structure learning. The success of this method will facilitate the rapid development of FLMs for different tasks in the future.

7 Conclusions

We have presented a data-driven approach to the selection of parameters determining the structure and performance of factored language models, a class of models which generalizes standard language models by including additional conditioning variables in a principled way. In addition to reductions in perplexity obtained by FLMs vs. standard language models, the data-driven model selection method further improved perplexity and outperformed both knowledge-based manual search and random search.

Acknowledgements

We would like to thank for Jeff Bilmes for providing and supporting the FLM software. This material is based upon work supported by the NSF and the CIA under NSF Grant No. IIS-0326276. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

Appendix: FLM Examples

The SRILM FLM file format indicates all three types of parameters required for specifying a given FLM. As a brief summary, we use the following bigram FLM specification for Arabic as an example. This FLM is the one found by the GA to give good perplexity results reported in section 5.

The first line of the FLM file specifies the initial set of conditioning factors. In this example,

```
W: 5 W(-1) P(-1) S(-1) R(-1) M(-1)
```

indicates that 5 conditioning factors $\{W_{-1}, P_{-1}, S_{-1}, R_{-1}, M_{-1}\}$ are used to estimate the current word W . The last number 28 indicates that 28 backoff paths are to follow. Each of the following 28 lines represents one backoff path and specifies information about the current conditioning factors, the factors available for backoff, and the smoothing option. For example, the line

```
W1,P1,S1,R1,M1      W1,S1,R1      kndiscount gtmmin 3 combine mean
```

indicates that $\{W_{-1}, P_{-1}, S_{-1}, R_{-1}, M_{-1}\}$ are the current conditioning factors and that $\{W_{-1}, S_{-1}, R_{-1}\}$ are the factors that can be individually dropped in the case when backoff is required. The remaining portion specifies the smoothing method for backoff. In this case, modified Kneser-Ney discounting (kndiscount) is used, minimum count required for including the n-gram is 3 (gtmmin), and the $g()$ function that combines three parallel backoff paths for dropping factors $\{W_{-1}, S_{-1}, R_{-1}\}$ are combined by taking their mean (combine mean). For all the following example FLM files, interpolate refers to the interpolation of the current node with the value returned by the backoff paths $g()$. The smoothing options ukndiscount, cdiscount, and wbdiscout refer to unmodified Kneser-Ney discounting, absolute discounting, and Witten-Bell discounting, respectively. Refer to [34] for a more detail explanation on FLM file specification.

The full set of factors for the Arabic FLM files are words (W), patterns (P), stems (S), roots (R), and morphological classes (M). The full factor set for Turkish FLMs are words (W), roots (R), part-of-speech (P), number (N), case (C), other information (O), and remaining factors (G). (See section 4).

The following is an example bigram FLM for Arabic discovered by GA search. Figure 9 shows the corresponding backoff graph.

```
W: 5 W(-1) P(-1) S(-1) R(-1) M(-1) arabic-bigram.count.gz arabic-bigram.lm.gz 28
```

W1,P1,S1,R1,M1	W1,S1,R1	kndiscount	gtmin 3	combine	mean	
P1,S1,R1,M1	P1,S1,R1,M1	kndiscount	gtmin 5	combine	mean	
W1,P1,R1,M1	W1,P1,R1,M1	ukndiscount	gtmin 2	combine	mean	
W1,P1,S1,M1	W1,P1,S1,M1	ukndiscount	gtmin 2	combine	mean	
S1,R1,M1	S1,R1,M1	ukndiscount	gtmin 2	combine	mean	
P1,R1,M1	P1,R1,M1	ukndiscount	gtmin 2	combine	mean	
P1,S1,M1	P1,S1,M1	ukndiscount	gtmin 2	combine	mean	
P1,S1,R1	P1,S1,R1	kndiscount	gtmin 5	combine	mean	
W1,R1,M1	W1,R1,M1	cdiscount	1	gtmin 4	combine	mean
W1,P1,M1	W1,P1,M1	kndiscount	gtmin 3	combine	mean	
W1,P1,R1	W1,P1,R1	kndiscount	gtmin 5	combine	mean	
W1,S1,M1	W1,S1,M1	kndiscount	gtmin 3	combine	mean	
W1,P1,S1	W1,P1,S1	ukndiscount	gtmin 2	combine	mean	
R1,M1	R1	ukndiscount	gtmin 2	interpolate		
S1,M1	S1	ukndiscount	gtmin 2	interpolate		
S1,R1	S1	ukndiscount	gtmin 2	interpolate		
P1,M1	P1	cdiscount	1	gtmin 4	interpolate	
P1,R1	P1	cdiscount	1	gtmin 4	interpolate	
P1,S1	P1	ukndiscount	gtmin 2	interpolate		
W1,M1	W1	wbdiscount	gtmin 1	interpolate		
W1,R1	W1	cdiscount	1	gtmin 4	interpolate	
W1,P1	W1	wbdiscount	gtmin 1	interpolate		
W1,S1	W1	cdiscount	1	gtmin 4	interpolate	
M1	M1	ukndiscount	gtmin 2	interpolate		
R1	R1	cdiscount	1	gtmin 4	interpolate	
S1	S1	wbdiscount	gtmin 1	interpolate		
P1	P1	cdiscount	1	gtmin 4	interpolate	
0	0	kndiscount	gtmin 1			

The following is an example bigram FLM for Turkish discovered by GA search. Figure 10 shows the corresponding backoff graph

```

W: 5 W(-1) P(-1) N(-1) R(-1) C(-1) turkish-bigram.count.gz turkish-bigram.lm.gz
15
W1P1N1R1C1 W1 kndiscount gtmin 1 interpolate
P1N1R1C1 P1N1R1C1 kndiscount gtmin 1000000 combine mean
N1R1C1 R1 kndiscount gtmin 1 interpolate
P1R1C1 R1 kndiscount gtmin 1 interpolate
P1N1C1 N1 kndiscount gtmin 1 interpolate
P1N1R1 N1 kndiscount gtmin 1 interpolate
N1C1 N1C1 kndiscount gtmin 1 combine mean
P1C1 P1C1 kndiscount gtmin 1 combine mean
P1R1 P1R1 kndiscount gtmin 1 combine mean
N1 N1 kndiscount gtmin 1 interpolate
C1 C1 kndiscount gtmin 1 interpolate
P1 P1 kndiscount gtmin 1 interpolate
R1 R1 kndiscount gtmin 1 interpolate
0 0 kndiscount gtmin 1

```

References

- [1] R. Alexrod. The evolution of strategies in the iterated prisoner's dilemma. In L. D. Davis, editor, *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann, 1987.

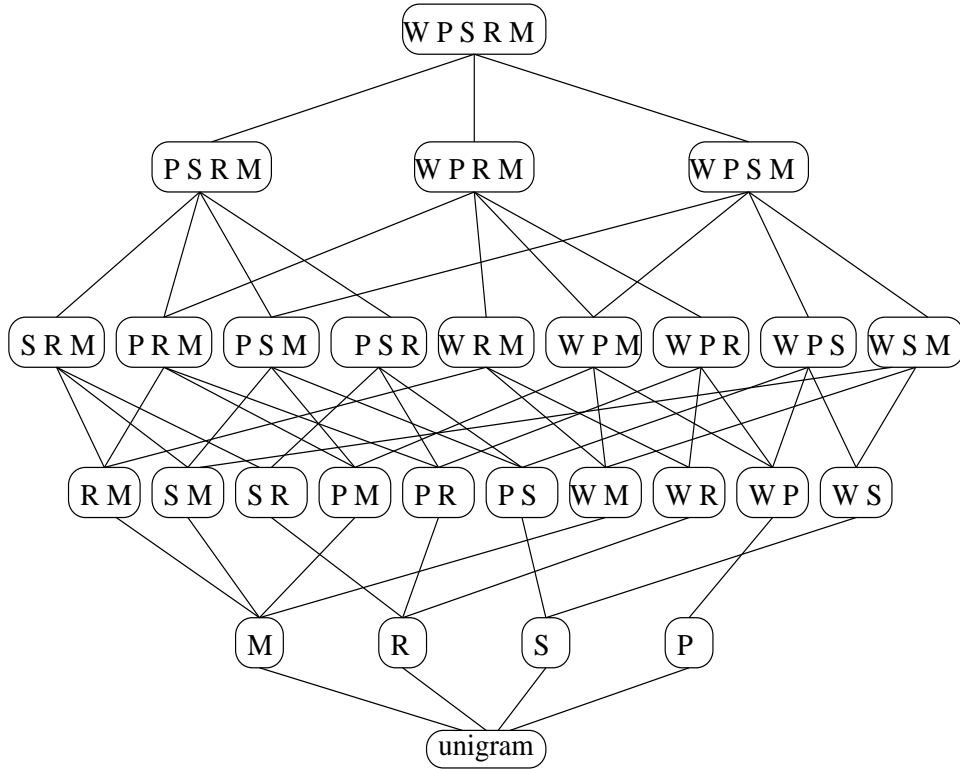


Figure 9: Backoff graph of the Arabic bigram specified in the FLM file. The characters W, P, S, R, M represents the previous word, pattern, stem, root, and morphological class factors, respectively.

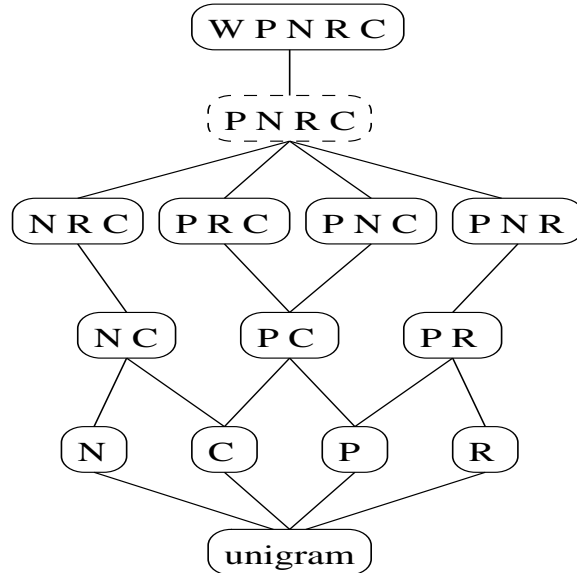


Figure 10: Backoff graph of Turkish bigram specified in the FLM file. The characters W, P, N, R, C represent the previous word, part-of-speech, number, root, and case, respectively. The dashed node corresponds to skipping a level. If not enough counts exist for $(W_t, W_{t-1}, P_{t-1}, N_{t-1}, R_{t-1}, C_{t-1})$ (root node), then both W_{t-1} and an additional factor are dropped. Due to the combine mean specification, the actual probability estimate will be the average of the estimates from $P(W_t|N_{t-1}, R_{t-1}, C_{t-1})$, $P(W_t|P_{t-1}, R_{t-1}, C_{t-1})$, $P(W_t|P_{t-1}, N_{t-1}, C_{t-1})$, and $P(W_t|P_{t-1}, N_{t-1}, R_{t-1})$.

- [2] R.K. Belew. Interposing an ontogenic model between genetic algorithms and neural networks. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing (NIPS5)*. Morgan Kaufmann, 1993.
- [3] Anja Belz. Discovering phonotactic finite-state automata by genetic search. In *Proceedings of COLING-ACL*, pages 1472–1474. Morgan Kaufman, 1998.
- [4] Jeff A. Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In *Proceedings of HLT/NAACL*, pages 4–6, 2003.
- [5] P.F. Brown et al. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.
- [6] B. Byrne, J. Hajic, P. Ircing, F. Jelinek, S. Khudanpur, P. Krbec, and J. Psutka. On large vocabulary continuous speech recognition of highly inflectional language - Czech. In *Proceedings of Eurospeech*, 2001.
- [7] K. Çarki, P. Geutner, and T. Schultz. Turkish LVCSR: towards better speech recognition for agglutinative languages. In *Proceedings of ICASSP*, 2000.
- [8] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University, 1998.
- [9] J.P. Cohoon and W.D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):956 – 964, November 1987.
- [10] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Erlbaum, 1985.
- [11] P. Dupont and R. Rosenfeld. Lattice based language models. Technical Report CMU-CS-97-173, Department of Computer Science, CMU, 1997.
- [12] J. Eggens, D. Feillet, S. Kehl, M. O. Wagner, and B. Yannou. Optimization of the keyboard arrangement problem using an ant colony algorithm. *European Journal of Operational Research*, 148:672–686, 2003.
- [13] Hsiao-Lan Fang, Peter Ross, and Dave Corne. A promising genetic algorithm approach to job-shop scheduling, re-scheduling, and open-shop scheduling problems. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 375–382, San Mateo, CA, 1993. Morgan Kaufmann.
- [14] S. Forrest and T. Jones. Modeling complex adaptive systems with Echo. In R.J. Sonier and X.H. Yu, editors, *Complex Systems: Mechanism of Adaption*. IOS Press, 1994.
- [15] Stephanie Forrest, Brenda Javornik, Robert E. Smith, and Alan S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1993.
- [16] William A. Gales. Good-Turing smoothing without tears. *Journal of Quantitative Linguistics*, 1995.
- [17] P. Geutner. Using morphology towards better large-vocabulary speech recognition systems. In *Proceedings of ICASSP*, pages 445–448, 1995.
- [18] D. Gildea. *Statistical Language Understanding Using Frame Semantics*. PhD thesis, University of California, Berkeley, 2001.

- [19] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(493–530), 1989.
- [20] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [21] F. Gruau. Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In L.D. Whitley and J.D. Schaffer, editors, *COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE Computer Society, 1992.
- [22] D. Hakkani-Tür, K. Oflazer, and Gökhan Tür. Statistical morphological disambiguation for agglutinative languages. In *Proceedings of COLING*, 2000.
- [23] D. Hakkani-Tür, K. Oflazer, and Gökhan Tür. Statistical morphological disambiguation for agglutinative languages. *Journal of Computers and Humanities*, 36(4), 2002.
- [24] W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Artificial Life*, volume II. Addison-Wesley, 1992.
- [25] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(495–502), 1987.
- [26] J. H. Holland. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 593–623. Morgan Kaufmann, Los Altos, 1986.
- [27] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [28] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.
- [29] Frederick Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397, Amsterdam, The Netherlands, May 1980.
- [30] Gand Ji and Jeff Bilmes. Multi-speaker language modeling. In *Proceedings of HLT/NAACL*, pages 137–140, 2004.
- [31] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975. Dissertation Abstracts International 36(10), 5140B (University Microfilms No. 76-9381).
- [32] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, March 1987.
- [33] D. Kiecza, T. Schultz, and A. Waibel. Data-driven determination of appropriate dictionary units for Korean LVCSR. In *Proceedings of ICASSP*, pages 323–327, 1999.
- [34] K. Kirchhoff et al. Novel speech recognition models for Arabic. Technical report, Johns Hopkins University, 2002.

- [35] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, pages 461–476, 1990.
- [36] Anne Kool. Literature survey. <http://citeseer.ist.psu.edu/kool99literature.html>.
- [37] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [38] S. Kwong, Q. H. He, K. W. Ku, T. M. Chan, K. F. Man, and K. S. Tang. A genetic classification error method for speech recognition. *Signal Processing*, 82(5):737–748, May 2002.
- [39] W. B. Langdon. Natural language text classification and filtering with trigrams and evolutionary NN classifiers. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 210–217, Las Vegas, Nevada, USA, 8 2000.
- [40] M. Lankhorst. Breeding grammars: Grammatical inference with a genetic algorithm. Technical Report Computer Science Report CS-R9401, University of Groningen, The Netherlands, 1994.
- [41] LDC. <http://www ldc.upenn.edu/Catalog/LDC99L22.html>, 1996.
- [42] Sven Martin, Jörg Liermann, and Hermann Ney. Algorithms for bigram and trigram word clustering. *Speech Communications*, 24:19–37, 1998.
- [43] K. Kirchhoff et al. Novel approaches to Arabic speech recognition: Report from 2002 Johns-Hopkins summer workshop. In *Proceedings of ICASSP*, pages I–344–I–347, 2003.
- [44] T.P. Meyer and N.H. Packard. Local forecasting of high-dimensional chaotic dynamics. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting*. Addison-Wesley, 1992.
- [45] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.
- [46] K. Oflazer. Dependency parsing with an extended finite state approach. In *Proceedings of the 37th ACL*, 1999.
- [47] Sonia Parendekar and Katrin Kirchhoff. Multi-stream language identification using data-driven dependency selection. In *Proceedings of ICASSP*, Hong Kong, 2003.
- [48] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
- [49] S. Schulze-Kremer. Genetic algorithms for protein tertiary structure prediction. In B. Manderick R. Männer, editor, *Parallel Problem Solving from Nature II*, pages 391–400, North Holland, 1992.
- [50] Andreas Stolcke. SRILM- an extensible language modeling toolkit. In *Proceedings of ICSLP*, pages 901–904, Denver, Colorad, September 2002.
- [51] W. Wang. Factorization of language models through backing off lattices. Computation and Language E-print Archive, oai:arXiv.org/cs/0305041, 2003.
- [52] E.W.D. Whittaker and P.C. Woodland. Particle-based language modeling. In *Proceedings of ICSLP*, Beijing, China, 2000.
- [53] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 35(3):1085–1094, July 1991.

- [54] I. Zitouni, O. Siohan, and C.-H. Lee. Hierarchical class n-gram language models: towards better estimation of unseen events in speech recognition. In *Proceedings of Eurospeech - Interspeech*, pages 237–240, 2003.