

Learning rule ranking by dynamic construction of context-free grammars using AND/OR graphs

Anna Corazza (†) and Louis ten Bosch (‡)

(†) Dept. of Information Technology
University of Milan, Italy
corazza@dti.unimi.it

(‡) A²RT, Dept. of Language and Speech
University of Nijmegen, the Netherlands
l.tenbosch@let.kun.nl

Abstract

This paper¹ discusses a novel approach for the construction of a context-free grammar based on a sequential processing of sentences. The construction of the grammar is based on a search algorithm for the minimum weight subgraph in an AND/OR graph. Aspects of optimality and robustness are discussed. The algorithm plays an essential role in a model for adaptive learning of probabilistic ordering. The novelty in the proposed model is the combination of well-established methods from two different disciplines: graph theory and statistics.

The set-up of this paper is mainly theoretical, and we follow a quite formal approach. There is a close link with Optimality Theory, one of the mainstream approaches in phonology, and with graph theory. The resulting techniques, however, can be applied in a more general domain.

1. Introduction

The characteristics of a speech signal can often be described in a functional way in terms of an *ordering* of properties. For example, the result of phonological rules will in general depend on the ordering in which these rules have been applied. Conversely, given the input for a set of rules, the resulting output of these rules may specify the ordering in which these rules have to be applied in order to obtain this particular result. In particular, an output string obtained by the application of a list of rewrite rules provides information about the ordering of these rules. So, the question arises to what extent statistical properties of a collection of strings can be adequately described by specifying the statistical ordering of the associated rules.

Often rules can be interpreted in two ways: (1) as actions, e.g. to be applied on an underlying (canonical) string, or (2) as constraints that are to be met. In this paper, we will focus on the second interpretation. In this interpretation, an input datum is checked against all constraints (requirements) according to a specific ordering. By evaluating each constraint one by one and checking its match with the input datum, the ranking index of the first violating constraint can be considered a measure for the match between datum and constraint list. In case of a set or stream of input data, the issue is how to dynamically learn

the optimal constraint ordering so as to maximize the 'match' between the data set and the specific ordered list of constraints.

This optimisation problem is closely related to the constraint ranking problems considered in Optimality Theory. Optimality Theory (see e.g. [1]) studies how constraints can be organized in a hierarchy or 'grammar' such that this hierarchy (e.g. a linear ranking) optimally reflects the structure in the input data, and how a listener can learn such a hierarchy. One of the first ranking algorithms, the 'recursive demotion' algorithm ([2]), iteratively adapts the ranking such that the least violated constraints become ranked highest, by evaluating the input against the list of constraints. The 'recursive demotion' algorithm is a hard-ranking algorithm which cannot deal with gradual violation or satisfaction of constraints. Soft-ranking models such as the Gradual Learning Algorithm (GLA, see e.g. [3]), show that the *probabilistic* ranking of constraints is a better alternative. Soft ranking is usually based on a continuous ranking scale. The basic idea is to associate each constraint 'x' with a real number $\text{pos}('x')$, such that $\text{pos}('a') < \text{pos}('b')$ means that 'a' is likely to outrank 'b'. The larger the distance between $\text{pos}('a')$ and $\text{pos}('b')$, the more likely 'a' occurs before 'b' (and reverse). In the GLA approach, each constraint is associated with a gaussian distribution, which makes it possible to explain variations in the input by one single ranking.

In [4], a probabilistic ranking algorithm (PRA) has been described that extends GLA in two directions. First, an analytical version of the probabilistic ranking algorithm was described. PRA uses the Borda index of a constraint and uses Expectation Maximisation ([5]) to find the statistically optimal constraint ranking. This analytical approach also holds for *partially* ordered set of constraints provided certain boundary conditions are met concerning the statistics in the ordering of constraints (see [4] and references therein). Second, it was proposed to use a Context-Free Grammar (CFG) prior to the probabilistic ranking step in order to increase the modelling power of PRA. However, the (adaptive) construction of an optimal CFG is a computationally hard problem. The remainder of this paper is devoted to an approximate solution for this problem.

We will first show how the use of a CFG leads to the adequate modelling of a larger group of orderings. Next, an algorithm will be described for the construction of a CFG based on an incoming stream of data. This construction of the CFG is based on the search for the optimal subgraph in a weighted so-called AND/OR-graph. Next, we present the precise construc-

¹This paper is the result of a joint cooperation in which the first author focussed on the construction of Context-Free Grammars, while the second author is responsible for the embedding in the probabilistic ranking framework and for the Brun's chain used in the heuristic search.

tion of the grammar, while the final section gives some conclusions and future work.

2. Probabilistic ranking and context-free grammars

In this section, we briefly describe the use of the Context-free Grammar (CFG, see below) to enhance the modelling power of the probabilistic ranking algorithm PRA. A more precise introduction to CFGs is presented in the next section. An example of a data set that can be modelled by PRA without any grammar is $S = \{ 'abc', 'bac', 'bca' \}$ (with $'b' < 'c'$ and floating $'a'$). This set cannot be modelled by GLA (for details see [4]). Examples of data sets that cannot be learned at all by PRA are $S = \{ 'abc, 'bca, 'cab' \}$ or $S = \{ 'abcd', 'cdab' \}$. Data sequences such as $S = \{ 'abcd', 'abcd', 'abdc', 'bacd', 'bacd', 'abcd', \dots \}$ leads to a ranking of four constraints $'a' < 'b' < 'c' < 'd'$. However, any ranking of these four constraints leads to a probability distribution on the set of 24 permutations of $'abcd'$, which may or may not be in accordance with the observed frequencies in the sequence. Whether a modelling is possible depends on this probability distribution (see also [6]).

The integration of a CFG with the ranking algorithm will lead to an increased modelling power and will be especially useful to factor out groups of constraints that behave as 'mutually independent'. For example, the set $S = \{ 'abcd', 'bacd', 'cdab', 'cdba' \}$ cannot be modelled by PRA. However, by using an appropriate grammar such as (A and B denoting non-terminals)

$$\begin{aligned} S &\rightarrow AB|BA \\ A &\rightarrow 'ab'|'ba' \\ B &\rightarrow 'cd' \end{aligned}$$

a modelling is possible by using the PRA to deal with the first rule. So, the aim is to construct a grammar such that rules involving permutations at the right-hand side can be modelled by PRA. In general, the CFG-based solution produces an eventual rule representation in the following form (capitals denoting non-terminals):

$$\begin{aligned} S &\rightarrow \text{prob. ranking}(X_1, X_2, \dots, X_k) \\ X_1 &\rightarrow \text{prob. ranking}(Y_1, \dots, Y_m) \\ &\vdots \\ Z_1 &\rightarrow \text{prob. ranking}('a'_1, \dots, 'a'_n) \\ &\vdots \end{aligned}$$

The learnability and robustness of the construction of the CFG is the focus of the next section.

3. The context-free grammar construction

In this section the static problem is considered: given a sequence of strings $S = (\sigma_i, i = 0, 1, \dots, N)$, find a CFG G deriving them. The input strings S are defined on a (finite) alphabet Σ . As in [7], the first step consists in constructing the *Suffix Tree* (ST) corresponding to S . The ST allows the identification of all repeated substrings, which can then be advantageously factorized by the grammar. The ST is then used as an implicit representation of an (alternating) AND/OR graph. It can be shown that there is a one-to-one mapping between CFGs and this kind of AND/OR graphs. The desired grammar is then built by finding an optimal weighted subgraph in the AND/OR graph.

In the following these steps are presented and exemplified by the case of input strings $babab$ and $abbab$.

3.1. The suffix tree

An ST [8] is a rooted directed tree associated to an input string σ . Each node is labeled with a substring in σ ; only the root corresponds to the empty string. The concatenation of the substrings corresponding to the nodes on each path from the root to a leaf node gives a suffix of σ . In this way, a one-to-one mapping is defined between leaf nodes and suffixes. Therefore the ST has exactly $|\sigma| + 1$ leaf nodes.

Each internal node has at least two children. The strings associated to children of the same node must differ at least by the first character. In this way, the search for a string in the tree is deterministic and the number of nodes in the tree is linear with respect to $|\sigma|$.

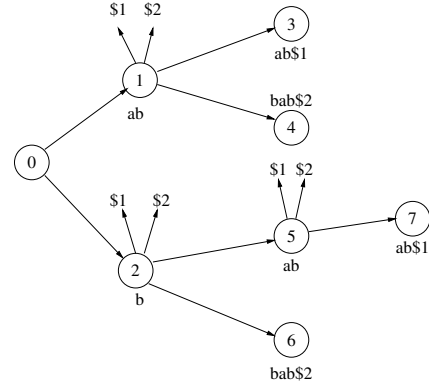


Figure 1: Example of ST for the input string $babab\$1abbab\2 .

In our case, the input string is given by the concatenation of the input strings in S , separated by delimiters $\$i$: $\sigma = \sigma_0\$0\sigma_1\$1 \dots \sigma_N\$N$, with $\$i \notin \Sigma$. In summary, delimiters are different from input characters and from one another, and strings associated to ST internal nodes are completely included in one of the input strings σ_i .

As only strings associated to leaf nodes can contain delimiters, they can be cut to the first delimiter without changing the results of the following processing. In conclusion, the ST will contain a path for each input string and for all their suffixes. Let us call C_T the set of the leaf nodes corresponding to complete input strings.

For the construction of the ST in linear time, the Ukkonnen algorithm is used [8], because of its incremental strategy which will be very useful in the following section.

3.2. Context-free grammars and AND/OR graph.

A Context-Free Grammar is characterized by a set of terminal symbols, a set of nonterminal symbols containing a start symbol S and a set of rewriting rules. In our case, the set of terminal symbols is given by $\Sigma \cup \{ \$i, 0 \leq i \leq N \}$.

The nonterminal symbols are defined by referring to the nodes in the ST. Each node potentially is a nonterminal symbol in the grammar: the language generated by the symbol is given by the only string associated to the node. Our goal is to decide which symbols must be introduced in order to build an optimal grammar.

The nonterminal set is initialized by considering all the leaf nodes n_i corresponding to the input strings ($n_i \in C_T$). In the example of Figure 1 two nonterminal symbols corresponding to nodes 7 and 4 are introduced at this point, namely N_7 and N_4 . In correspondence of each of these symbols, a new rule $S \rightarrow n_i$

is introduced, to add the corresponding string to the language. Starting from them, we must decide which nonterminal symbols and which rules rewriting them are included in the grammar.

Given a nonterminal symbol, a number of new rules are possible to rewrite it, depending on the position of the corresponding node in the ST. If the parent of the node is the root, the node is rewritten by its associated string, e.g. $N_1 \rightarrow ab$ or $N_2 \rightarrow b$. Otherwise, a nonterminal binary rule is possible for each of the nodes on the path from the root to the node. The right-hand side of the rule includes this node and the node corresponding to the complement of the string. The choice of the node implies the choice of the best split of the input string. For example, $N_7 \rightarrow N_2 N_3$ corresponds to splitting the string $babab$ into $b abab$, while the alternative split $N_7 \rightarrow N_5 N_1$ corresponds to $bab ab$.

A one-to-one mapping can be defined between CFGs and alternating AND/OR graphs [9]. An AND/OR graph [10] is a directed acyclic graph $G = (V, A)$ with a single source vertex $s \in V$ having in-degree 0 (Figure 2). Each vertex is either an and-vertex or an or-vertex. In alternating AND/OR-graphs, all children² of and-vertices are or-vertices and vice-versa.

A solution is a subgraph of the AND/OR-graph. It includes: the source node; at least one child for each or-vertex in the solution; and all children for each and-vertex. Each solution of the AND/OR graph corresponds to an acceptable grammar.

Each symbol of the CFG corresponds to an or-vertex. The source vertex corresponds to the start symbol of the grammar. Each nonterminal symbol has a child for every rule rewriting it. Vertices corresponding to rules are and-vertices, having children corresponding to all nonterminals in the right-hand side of the rule. In this work, the order in the right-hand side symbols is unimportant, and will be disregarded. Moreover, only nonterminal productions are considered. As the corresponding language is finite, the grammars we consider are non-recursive, and therefore the graph is acyclic.

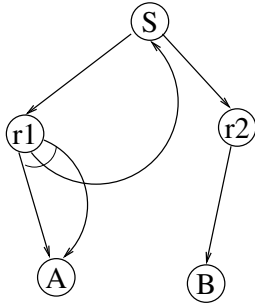


Figure 2: AND/OR-graph corresponding to the grammar $S \rightarrow A S A \mid B$. Note that the cycle corresponds to the recursive rule $S \rightarrow A S A$.

The ST gives an implicit representation of the AND/OR graph corresponding to the grammar. The following section will explain how the interesting parts of the AND/OR graph are expanded.

In order to compare different solutions to find the optimal one, we consider weighted AND/OR graphs, where a cost is associated to each edge. A weight can then be associated to each solution as the sum of the weights associated to each involved

²As the graph is directed and acyclic, we use the term *child* to indicate the successor of a node, and the term *parent* to refer to the predecessor.

edge. Different cost functions can be used to bias the solution in the preferred direction. For example, we can associate cost one to each edge leaving an or-vertex, and a null cost to all the others. Additional vertices can be added after each terminal vertex, so that nonterminal symbols corresponding to terminal rules can also be taken into account. In this way, the minimum weight solution corresponds to a grammar with the minimum number of nonterminal symbols.

By choosing a convenient cost function, the problem becomes the search for the minimum weight AND/OR graph solution, which is known to be NP-complete [10]. A number of heuristic solutions are proposed in the literature [11], which get polynomial complexity in case of additive scores.

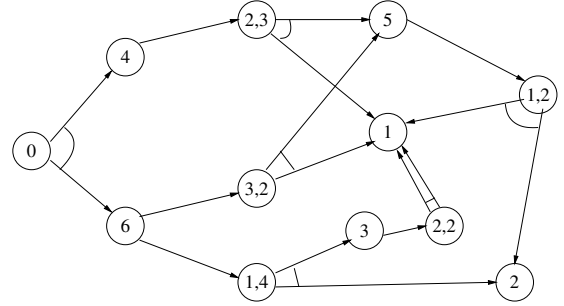


Figure 3: Complete AND/OR-graphs for the ST in Figure 1; note that the AND/OR-graph is left implicit in the implementation.

3.3. Heuristic search

The current search strategy is guided by a heuristic function. The solution is built starting from the source vertex and expanding all children of all and-vertices. Decisions must be taken on or-vertices, for which just one child must be chosen. This decision is taken without exploring all the solution graph, but using a heuristic function which gives an optimistic estimate of the cost.

At each or-vertex, only the length of the considered string is known. We will use a heuristic estimate of a lower-bound of the cost function associated with that vertex. This heuristic is based on the hypothesis that the string is optimal for compression, that is, based on repetitions of the same character. In this hypothetical case, the minimum number of nodes required is at most equal to the length of specific so-called addition chain [12]. An addition chain of a set of integers $\{a_j\}$ is a set of integers $\{b_k\}$ such that each element in the set $\{a_j\}$ is the sum of at most two elements from the set $\{b_k\}$. Each addition chain corresponding to the set $\{N\}$ is uniquely related to a CFG for the one-character sentence a^N . Brun's algorithm (see [12]) is one of the algorithms to approximate the shortest possible addition chain. For a single positive integer N , this algorithm yields the chain $\{a_j\}$ with $a_1 = N$, $a_{i+1} = a_i/2$ if a_i even and $a_{i+1} = a_i - 1$ if a_i odd. Let $b(N)$ denote the length of this chain for N .

Since each and-vertex corresponds to a rewriting rule, it therefore corresponds to a split of the current substring in two fragments, respectively of length i and j , denoted by (i, j) . The minimum cost derived by this split is given by the number of vertices introduced by the larger of the two substrings in case it is a^i (if $i \geq j$): $b(i)$. It corresponds to the case in which all the vertices necessary to expand the other part of the string are

included in the ones generated to expand a^i .

In conclusion, a heuristic score is associated to each and-vertex (split). If the split is (i, j) , the score is given by the length of the Brun's chain of the greatest of the two. This gives the minimum possible number of vertices to be introduced to implement that split. If a split under examination has a Brun score which exceeds the cost of the tree subgraph expanded so far, then it can be pruned by the search.

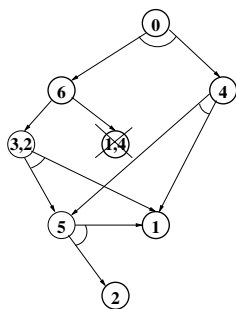


Figure 4: Search on the AND/OR-graph: the unique split node under 5 is omitted for simplicity.

Let us consider the general idea by using the expansion of node 6 in the ST depicted in Figure 4. It has two possible expansions: $(3, 2)$ and $(1, 4)$. Both have Brun score 3, that is, when expanded will have a cost which is greater or equal to 3. Let us start with $(3, 2)$, which corresponds to the first two elements of the Brun's chain. Its expansion gives cost 3. The heuristic function tells us that split $(1, 4)$ needs not be expanded as it can not be better than split $(3, 2)$, at least at this point of the computation.

3.4. Incremental construction

In this section we consider the processing necessary when a new string is added to the set to be represented. As a first step, it is concatenated to the input string, together with a new delimiter $\$_{N+1}$: $\sigma = \sigma_0 \$_0 \dots \sigma_N \$_N \sigma_{N+1} \$_{N+1}$.

The Ukkonen's algorithm is used to build the ST. It considers the input string from left to right, starting with the empty prefix and considering prefixes of increasing length. Some of the ST obtained during the process can be *implicit*, that is, the number of leaf nodes can be lower than the length of the current prefix. However, at the end of each string, as the delimiter is different from any other character, for the current ST all the considered properties hold [8].

During the ST update corresponding to the addition of a new input string, at least the leaf node corresponding to it is added to the tree (and to the set corresponding to complete strings, C_T). Moreover, other internal and leaf nodes can be added, respectively corresponding to new common substrings and to new suffixes.

In correspondence, new nonterminal symbols and new rules can be considered to find the optimal grammar. Therefore, these new nodes must be included in the search. The new node in C_T corresponds to a new source vertex child. As the source is an and-vertex, this node must be mandatorily expanded.

No other leaf nodes need to be considered. The problem is more subtle for the other internal nodes, which can interfere with the expansion of other or-vertices, and therefore with the global solution.

4. Discussion and conclusions

A novel approach to build a CFG from a sequence of sentences has been proposed. In this paper, it has been presented in the framework aiming at an efficient and adequate ordering of properties representing speech signal characteristics. The paper has a theoretical character, and the followed approach can be applied in a more general domain.

The construction of the CFG is based on the search for the optimal subgraph in an AND/OR-graph. The main search issue is the proper treatment of the non-additive property of the costs along the paths in the AND/OR-graph. The current implementation of the search is guided by an heuristic function based on the optimistic case in which the corresponding substrings are repetitions of the same character. Future research focusses on a better estimation of the costs associated to the arcs in the AND/OR-graph. Well-established numerical optimization methods such as simulated annealing will be investigated in the follow-up of this research.

5. References

- [1] R. Kager, *Optimality Theory*. Cambridge University Press, 1999.
- [2] B. Tesar and P. Smolensky, "Learnability in optimality theory," *Linguistic Inquiry*, vol. 29, pp. 229–268, 1998.
- [3] P. Boersma and H. Bruce, "Empirical tests of the gradual learning algorithm," *Linguistic Inquiry*, vol. 32, pp. 45–86, 2001.
- [4] L. Ten Bosch, "Probabilistic ordering of constraints," in *Proceedings of ICSLP-02*, (Denver, Co, USA), pp. 2289–2292, 2002.
- [5] C. Lee and J. Gauvain, "Bayesian adaptive learning and MAP estimation of HMM," Chapter 4 in: *Automatic Speech and Speaker Recognition*, C.H. Lee et al., eds., Kluwer Academic Publishers, 1996.
- [6] L. Ten Bosch, "On the probabilistic ordering of constraints," in *Proceedings of the Institute of Phonetic Sciences*, vol. 24, (University of Amsterdam), 2002.
- [7] A. Corazza and A. Lavelli, "An n -best representation for bidirectional parsing strategies," in *Proc. of AAAI-94 Workshop on the Integration of Natural Language and Speech Processing*, (Seattle, Washington, USA), pp. 7–14, July 1994.
- [8] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, USA: Cambridge University Press, 1997.
- [9] P. Hall, "Equivalence Between AND/OR Graphs and Context-Free Grammars," *Communications of the ACM*, vol. 16, no. 7, pp. 444–445, 1973.
- [10] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, USA: W.H. Freeman and Company, 1979.
- [11] A. Mahanti and A. Bagchi, "AND/OR Graph Heuristic Search Methods," *Journal of ACM*, vol. 32, no. 1, pp. 28–51, 1985.
- [12] J. Bos and M. Coster, "Addition chain heuristics". *Proceedings of Crypto '89*, pp. 400–407, 1990.