INTERACTIVE GRAMMAR INFERENCE WITH FINITE STATE TRANSDUCERS

Sasha P. Caskey, Ezra Story, Roberto Pieraccini

Speechworks International
55 Broad Street, New York, NY 1004 USA
{scaskey, ezra, roberto}@speechworks.com

ABSTRACT

We propose here a method for improving the coverage of handcrafted context free grammars based on a set of new sentence examples. The described algorithm aims at finding the minimal set of modifications to the grammar that increase its coverage while preserving its original structure. The algorithm is based on a Finite State Transducer (FST) representation of context free grammars. The inference method includes an interactive component that allow developers to control the generalization of the new grammar.

1. INTRODUCTION

Most commercial dialog systems are based on hand crafted context free grammars (CFGs) that, after usability tests and during the initial stages of deployment, need to be maintained and modified by the developer in order to improve their coverage of the language [1]. Since language data (e.g. a corpus of transcribed utterances) is typically not available during the design and development phases of a dialog system, the initial grammars are created by hand with an attempt at giving a reasonable coverage. After a series of usability tests and during early deployment, transcriptions of the input utterances are becoming available periodically, a portion of which are not covered by the initial grammars. It is the job of the developer to analyze the transcriptions and modify the grammars accordingly. As the number of deployed applications and the number and size of grammars used in those applications grows, so do the development and maintenance costs.

Speech application developers and user interface specialists are trained to build grammars with reasonable generalization capabilities. For example, given the sentence "I'd like to go to Boston", a speech application developer would produce a grammar that would also cover sentences with similar semantic structure, such as "I want to go to New York", or "I am going to San Francisco". A very simple grammar that covers these sentences is shown in Table 1 (with the symbol "|" representing disjunction). However, this grammar does not have any generalization capability, i.e. it will not be able to cover other sentences than those belonging to the original set. For instance, the grammar of Table 1 will not cover slight variations obtained by combinatorial permutations of the words and phrases in the original set, such as "I want to go to Boston". Generalization is usually achieved by giving structure to the grammar based on the developer's insight of the task, the user interface, and the underlying language. Structure is obtained, in CFG formalism, by arranging the grammar rules into a set of non-terminals which allow for a modular design of the language. For instance, for the above examples, one could think of a set of terminals such as MOVE, DEST and CITY. The grammar shown in Table 2, based on these non-terminal rules, allows for a good degree of generalization and covers a larger number of variations of the original set

```
S =  (I'd like to go to Boston |
       I want to go to New York |
       I am going to San Francisco)
```

Table 1: Example of an unstructured CFG grammar

```
S =  MOVE DEST
MOVE = (I'd like to go | I want to go | I am going)
DEST = to CITY
CITY = (Boston|New York|San Francisco,...)
```

Table 2: Example of a structured CFG

of sentences.

The task of creating and improving grammars from observed data can be automated with the help of automatic grammar inference algorithms. Many of these algorithms exist in the literature based on a purely statistical representation of formal languages, such as the inside-outside algorithm [2][3] , or on the combination of structured and stochastic models [4]. Nevertheless, statistically based grammar inference algorithms are generally very slow and require very large amounts of data, possibly annotated. In contrast, after usability test or early deployment, developers can rely only on a small set of sentences, generally on the order of a few hundred.

Some structured grammar inference algorithms, such as the Error Correcting Grammatical Inference (ECGI) algorithm [5] are more parsimonious and can be used to infer finite state grammars able to generalize over a set of examples. However, one of the drawbacks with automatically inferred grammars of this kind, is that they do not have structure, thus are hardly human readable, hence they are not easy to be modified and improved by hand. In this context it must be pointed out that hand tuning of applications is still a common procedure for improving the performance of commercial dialog systems.

Another issue with automatic inference of natural language grammars is generalization. Most of the learning algorithms tend to generalize in an uncontrolled way, partly because of the difficulty in providing the learning algorithm with a set of negative examples that would control over-generalization [6]. Thus, inferred grammars generally over-generate with respect to the underlying language, a property that is not desirable because it may reduce the constraining power of the speech recognizer, hence its accuracy.

We should also take into consideration that grammars are used in dialog systems not only for constraining the speech recognizer, but also for producing utterance interpretations. The common way of generating interpretation consists in augmenting the grammar with scripts (consisting of variable assignments in the simplest case) that would generate sets of key-value pairs from the parse tree. The use of scripts (typically in ECMA script languages, such as JavaScript, or a subset of them) for extracting semantic information is supported by a W3C candidate recommendation [7] and adopted today by most commercial speech recognizers. Thus, when new rules are added to the grammar, developers need to make sure that the scripts that generate the semantic information are still valid, possibly modifying them in order to account for the new rules. This argument contributes to the motivations for investigating supervised methods apt at increasing the coverage of grammars over a set of new examples, with the developer part of the training loop and controlling the process of grammar learning.

The method we will describe in this paper attempts at finding and suggesting a set of changes to the original grammar that would increase its coverage over a set of sample transcriptions, while maintaining the context free structure initially created by the developer. The developer would have the choice of accepting or rejecting those changes. The proposed method would thus allow the use of automatic grammar inference for an optimal modification of the grammar based on an imposed non-terminal structure, while leaving the choice to developers of modifying the grammar according to their understanding of the domain.

2. THE INFERENCE ALGORITHM

The problem of improving the grammar coverage of an existing CFG can be formalized as follows. Given a context free grammar G and an input sentence S that is not accepted by G, we want to find the minimal set of modifications to the grammar that would allow it to accept S. The *minimal set of modifications* is intended in terms of minimal edit (or Levenshtein) distance between the original grammar G and the improved grammar G', and it is defined as the number of substitutions, insertions and deletions of terminals performed on the rules of G in order to obtain G'.

In order to apply conventional search techniques for finding the minimal set of edits, it is convenient to convert the standard SRGS [7] grammar into an equivalent finite state transducer (FST) [8][9]. Notice that this operation is possible only if the original CFG can be represented by finite state machine, i.e. it does not have CFG rules with embedded recursion; most commercial recognizer explicitly forbid rules with embedded recursion.

This transformation of a CFG to a FST generally does not preserve the original non-terminal structure. In order to include and preserve non-terminal information into the FST we need to add nodes and arcs with the purpose of non terminal, or rule boundary labels. Labels can be included in an FST by adding an arc with the null symbol as input and the label in question as output. The label will be then propagated to any FST resulting from an arbitrary composition operation with the current FST.

An example of FST representation of a CFG with rule boundary labeling is shown in Figure 1, where the CITY portion of the CFG of Table 2 is represented. The network described in Figure 1 follows the usual conventions for FSTs, i.e. the arcs are labeled with *input:ouput* symbols, with ε representing the null symbol. The arc labeled as ε:{ represent the left boundary of the rule, while the arc labeled as ε:}CITY represent the right boundary and the name of the rule. The above described FST (F_G) includes thus all the structural information of the original CFG G.

In order to search for the minimal edit that would allow to parse the new sentence S, we will augment F_G with arcs and nodes that would represent all possible edit

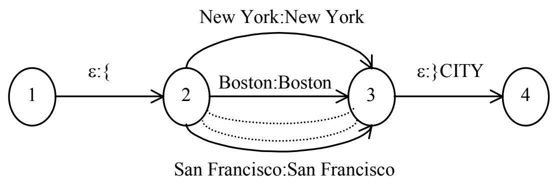

Figure 1: FST representation of rule CITY with rule boundary labels

operations. The augmented *inference* FST will be denoted by F_I . However, we need to maintain a mapping between the new arcs and nodes of F_I, which correspond to edit operations, and the original F_G, in order to be able to apply the final modifications to G. This is done by augmenting F_I with label arcs with the intent of marking the original state labels of F_G. Doing so we will have, in F_I, the necessary information that allows us to map any selected modification of F_I to the corresponding nodes of F_G (and then to G).

If any word appears in S that is not accounted for by the original grammar's set of terminals, the word is added to the FST vocabulary.

As we discussed earlier, F_I represents all the possible strings that can be obtained from F_G by arbitrary substitution, deletion, or insertion of terminal symbols. All the edit operations are restricted to within the rule boundaries, that is to say, no insertions are allowed across rules.

Let's define as *terminal* arc an arc of F_I whose input/output symbols are not labels, but rather terminal symbols of the corresponding CFG grammar. Let's also define as *terminal* node any node of F_I connected, either to the left or to the right, to at least one terminal arc. Insertions, in the resulting F_I, are thus accounted for by adding, to each terminal node, a number of arcs equal to the number of words in the dictionary. Each arc would have input/output symbols corresponding to each of all the possible words. This is represented symbolically in Figure 2 by the loop arcs labeled by Σ:Σ (where Σ is the set of all possible words). Deletion and substitution of words is accounted for by inserting a number of arcs equal to the number of words in the vocabulary, including the null symbol ε, in parallel to any terminal arc. The resulting FST would match any string obtained by G by applying an arbitrary number of the above defined edit operations. Since, for suggesting modifications to G, we are interested in knowing if an arc corresponds to an insertion, deletion, or substitution (with respect to the original grammar), we then label the modification arcs accordingly (i.e. we add *label* arcs to the leftmost connection of every insertion, substitution and deletion arc.) Also, for computing the edit distance, we associate a weight $w=1$ to each insertion, substitution and deletion arc (while the arcs of the original grammar have weight $w=0$). An example of the

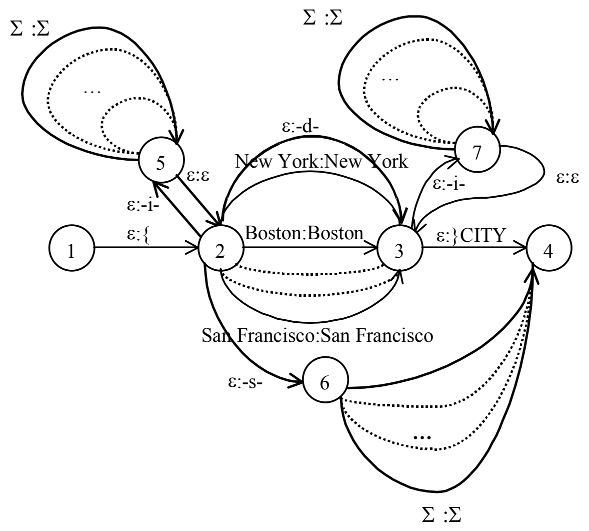

Figure 2: FST resulting from processing rule CITY through inference algorithm. Σ is the set of all terms (state labels are omitted for simplicity.)

resulting transducer is shown in Figure 2 for the non-terminal rule CITY of the sample grammar (state labels are omitted for clarity).

Thus F_I would represent strings such as :

_1 { 2 { _3 **I** _5 **want** _7 -i **Boston** _12 **to** _13 -d _15 }MOVE _16 { _22 **to** _25 { _26 -s **like** _31 }CITY _48 }DEST _56 }S

which corresponds to the insertion (denoted by the preceding symbol -i) of the word *Boston* after the word *want* and the deletion (denoted by -d) of a word at the end of the non-terminal MOVE, and the substitution (denoted by -s) of the word *like* for a word of the non-terminal CITY (the cost of this edit is 3). It has to be noted that all possible edits of the strings corresponding to the original grammar are allowed by F_I, producing thus lots of nonsensical sentences such as the one of the previous example. The symbols denoted by _N in the example (with N being a number) correspond to state labels. Thus, the string of the previous example carries the information that the insertion of the word *Boston* is located between the nodes labeled as 7 and 12 in the original F_G, as well as the deletion is between the nodes labeled as 13 and 15 and the substitution with the word *like* takes place between nodes 26 and 31 . Hence, if we wanted the original grammar to represent the (nonsensical) sentence *I want Boston to to like*, we would need to add a terminal arc labeled with the word *Boston* between the nodes 7 and 12 of F_G. Similarly we need to add an ε arc between the nodes 13 and 15, and an arc with the word *like* between nodes 26 and 31.

The next step of the inference algorithm thus consists in finding all the strings of F_I that would match the sequence of words in the new sentence S. To that purpose, an FST F_S that represents the input sentence S is then constructed and successively composed with the inference grammar, resulting in a composed FST F_C. F_C represents all the possible modification of G that would produce the sentence S. Since we are interested in the minimal set of modifications, an n-best search for the shortest path (where the length of a path corresponds to the sum of the weights on the arcs) is conducted on F_C. Typically the n-best return a set of equally scoring minimal edits, since the same sentence can be accounted for by different edits (e.g. a sequence substitution-insertion can also be realized as an insertion-substitution with the same partial score of 2).

For example, given the sentence "I would like to fly to Denver please" (note that *Denver* is not included in the original vocabulary), the n-best search on F_C returns the set of strings shown in Table 3 (state labels are not shown for clarity).

All the resulting strings in Table 3 have the same score (corresponding to the minimal length of 5 edits) and represent equally reasonable modifications, although some of them may not be semantically consistent with the non-terminal structure and may lead to wrong generalization. Developer should choose the set of modifications (i.e. the string) that lead to the most reasonable generalization.

However, the n-best strings resulting from the composed automaton F_C are hard to read and result from combinatorial expansion of independent modifications, in other words they present the information in a redundant manner. In order to present developers with a more effective view of the suggested modifications and allow them to choose the best independent ones, a more compact representation of the strings need to be attained. An example of compact representation of the edits of Table 3 is shown in Table 4. Here we divide the changes into independent groups, and within each group we assign each change to the non-terminal it pertains.

In order to obtain a compact representation, such as the one of Table 3, all the n-best strings of F_C with the top score are selected and arranged into an FST that is then determinized and minimized [10]. Then we need to determine the groups of *independent* modifications, i.e. the portions of the resulting network that group sets of independent alternative changes. For example, the alternative changes grouped under GROUP 1 in Table 3 affect sets of non terminals that do not intersect with the terminal affected by the changes in other groups (e.g. GROUP 2 in Table 3). If independent groups of modifications exist, the FST resulting from the previous step include *independent* nodes, i.e. nodes whose removal would produce disjoint sub-graphs.

Each independent sub-graph represents a set of alternative modifications to the original rules that are presented to the developer as in Table 4.

The labels included in the n-best FST, i.e. state and rule identifiers, rule bracketing, and the information on whether each terminal is derived from the original rule expansion, or is considered as an insertion, deletion, or substitution, are used for transforming the information derived from the n-best paths into modifications of the original grammar. In particular, rule, state id, and rule bracketing are used to determine which rule and where in the rule expansion the modification is located. Then, for each insertion a new optional terminal element is accounted for in the rule, for each deletion the corresponding terminal is made optional, and for each substitution a new terminal element is inserted as an alternative to the terminal it is substituting. The modified F_G is then converted back into SRGS format to be used by the recognizer.

It can be noticed that, although all modifications proposed by Table 4 are actually providing coverage for the new sentence, some of them would make the modified grammar over-generalize to incorrect sentences. For instance, among all the choices of group 1, only choice number 3 does not generalize to incorrect sentences, while, for instance, choice 1 generalizes incorrectly to phrases such as " I'd would like to go". Here is where developers could use their judgment and select the better choice for modifying the grammar.

{ { _s **I** _i **would like to** _s **fly** }MOVE { **to** { -s **Denver** -i **please** }CITY }DEST }S

{ { -i **I** -s **would like to** -s **fly** }MOVE { **to** { -s **Denver** -i **please** }CITY }DEST }S

{ { **I** -s **would** -i **like to** -s **fly** }MOVE { **to** { -s **Denver** -i **please** }CITY }DEST }S

{ { **I** -i **would** -s **like to** -s **fly** }MOVE { **to** { -s **Denver** -i **please** }CITY }DEST }S

{ { -s **I** -i **would like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY -i **please** }DEST }S

{ { -i **I** -s **would like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY -i **please** }DEST }S

{ { **I** -s **would** -i **like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY }-i **please** }DEST }S

{ { **I** -i **would** -s **like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY -i **please** }DEST }S

{ { -s **I** -i **would like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY }DEST -i **please** }S

{ { -i **I** -s **would like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY }DEST -i **please** }S

{ { **I** -s **would** -i **like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY }DEST -i **please** }S

{ { **I** -i **would** -s **like to** -s **fly** }MOVE { **to** { -s **Denver** }CITY }DEST -i **please** }S

Table 3: n-best edits with the top score.

GROUP	CHOICE	RULE	
1	1	MOVE	(I'd | **I) [would**]) like to (go | **fly**)
	2	MOVE	[**I**] (I'd | **would**) like to (go | **fly**)
	3	MOVE	I (want | **would like)** to (go | **fly**)
	4	MOVE	I [**would]** (want | **like**) to (go | **fly**)
2	1	CITY	(Boston | New York | San Francisco | **Denver**) **[please]**
	2	CITY	(Boston | New York | San Francisco | **Denver**)
		DEST	to $CITY **[please]**
	3	CITY	(Boston | New York | San Francisco | **Denver**)
		S	$MOVE $DEST **[please]**

Table 4: Compact representation of grammar edits.

As another example of the usefulness of human supervision in the inference process, consider the second group of modifications in Table 4. The three alternatives of Group 2 from Table 3 are equivalent, and none of them, selected, would over-generalize to incorrect sentences. The difference between the three choices is in the placement of the word *please* alternatively at the end of the expansion of the rule CITY, DEST or S. However a skilled grammar writer would probably select the third choice, which places the word *please* as a terminal of the root rule, since the word *please* does not belong syntactically or semantically to neither CITY nor DEST.Once the developer accepts a modification, the original grammar is modified accordingly.

3. CONCLUSIONS

A method is presented here for the improvement of handcrafted context free grammars over a set of examples outside the current grammar coverage. The method computes the set of minimal edits with respect to the original grammar which will account for the new sentences. and presents them in a compact and readable manner. Developers can choose among equivalent edits in order to get a better generalization of the grammar.

The interactive nature of the tool makes a formal evaluation difficult. It is certainly able to increase the grammar coverage as well as other inference algorithms, given the same amount of data. Moreover, maintaining the original structure makes the resulting grammar amenable to successive handcrafting and tuning by the developer. The order of presentation of the rule modifications to the developer can be made in such a way that the modification that increase coverage most (i.e. number of sentences covered by each modification) would appear first, helping improve the effectiveness of the tool. The interactive grammar inference tool described in this paper can be thought of as a basic component of a more sophisticated grammar editing and tuning tool..

4. ACKNOWLEDGMENTS

This work has been partially funded by the Commission of the European Communities under FASiL (Flexible and Adaptive Spoken Language and Multi-Modal Interfaces - http://www.fasil.co.uk/), contract number IST-2001-38685.

5. REFERENCES

[1] Barnard, E., Halberstadt, A., Kotelly, C., Phillips, M., "A Consistent Approach To Designing Spoken-dialog Systems," in *Proceedings of the Automatic Speech Recognition and Understanding Workshop*, Keystone, Colorado, December 1999.

[2] Baker, J.K, "Trainable grammars for speech recognition," in J.J., Wolf and D. Klatt, editors, Speech communication papers presented at the *97th Meeting of the Acoustical Society of America*, MIT, Cambridge, MA, June 1979

[3] Pereira, F., Schabes, Y, "Inside-Outside reestimation from partially bracketed corpora," in Proc. of the *Annual Meeting of the ACL*, pp 128-135, 1992

[4] Chelba, C., Jelinek, F., "Structured Language Modeling," in *Computer Speech and Language*, 14(4), pp 283-332, October 2000.

[5] Vidal, E, "Grammatical inference: An introductory survey," In R. C. Carrasco and J. Oncina, editors, Grammatical Inference and Applications, Proc. of *2nd ICGI, volume 862 of Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1994.

[6] Gold, E., M., "Language Identification in the limit," In *Information and Control,* 10(5), pp 447-474, 1967

[7] W3C Candidate Recommendation: Speech Recognition Grammar Specification, Version 1.0, 26 June, 2002 - http://www.w3.org/TR/speech-grammar/

[8] Mohri, M., Pereira, F. C. N., Riley, M,. "Weighted Finite-State Transducers in Speech Recognition," *Computer Speech and Language*, 16(1):69-88, 2002

[9] Schalkwyk, J., Hetherington L., Story, E., "Speech Recognition with Dynamic Grammars Using Finite State Transducers," ," in *Proceedings of Eurospeech 2003*., Geneva (Switzerland), September 2003.

[10] Mohri. M., Minimization Algorithms for Sequential Transducers. in *Theoretical Computer Science*, 234:177-201, March 2000.