CONSTRUCTING SMALL LANGUAGE MODELS FROM GRAMMARS

Francis Picard[1,2], *Dominique Boucher*[1] *and Guy Lapalme*

DIRO, Université de Montréal, C.P. 6128, Succ. Centre-ville, Montréal, Canada H3C 3J7

ABSTRACT

This paper presents a method for constructing small word graphs from regular grammars in a way to reduce the number of vertices in the resulting graph. Our method works at the grammar level instead of intermediate forms like finite automata. It represents a prime alternative to exact minimization algorithms, and is distinguished by its simplicity, its flexibility and by the fact that it avoids the determinization of the resulting graph or automaton.

1. INTRODUCTION

In the field of speech recognition, there is a constant trade-off between (1) recognition time and language model size and (2) recognition accuracy. This puts a strong emphasis on the development of language models leading to accurate recognition while being as efficient as possible. Various techniques have been developed to optimally compress the language models without giving rise to a degradation in the recognition performance. These techniques are usually independant of the way these language models were obtained. For example, [1] and [2] develop reduction methods for lexical trees, while the authors of [3] propose similar algorithms for automata, and those of [4] and [5] for probabilistic automata.

In the InfoSpace speech recognition system, regular grammars are specified using a proprietary grammar specification language (GSL). In order to be used by the recognition engine, these grammars are translated (compiled) to word graphs. In a previous version of the system, no particular effort was done to construct small word graphs. Even medium-sized grammars (a few hundred lines) led to very large word graphs, requiring a lot of memory and slowing down the recognition process considerably.

Since word graphs are very similar to finite-state automata, an obvious solution might have been to use a determinization algorithm followed by a minimization algorithm on the graph obtained by the compilation method, with possible graph/automaton conversions.

This approach has the disadvantage of building an oversized graph only to reduce it afterwards. This process is too time- and memory-consuming in a real-time setting, i.e. if grammars are to be compiled on-the-fly at application run time. A better approach would be to periodically apply the determinization/minimization processes during the construction of corresponding automaton. In the end, the automaton would have to be converted into a word graph.

We rather chose to work at the grammar level, attempting to eliminate much of the redundancy contained in the grammar during the first stage of its conversion into a word graph. We will see that the heuristic method described here, which we call expansion-compression, represents a viable alternative to determinization/minimization, not only because of the great reduction in size of the resulting word graphs, but also for its increased flexibility. It also offers several advantages that can be quite interesting, depending on the context of application. This method is a general one and can be applied to the compilation of grammars towards all variants of automata or graphs.

The paper is organized as follows. The remaining of this section defines the basic concepts and presents the basic expansion algorithm. Section 2 explains the general method and describes the implementation. Experimental results are then given in section 3 and a discussion follows in section 4.

1.1. Definitions

A grammar is defined by a finite set V of non-terminals (or variables), a finite set T of terminals (or vocabulary words), and a set of *production rules* of the form $A \rightarrow \alpha$, where A (the LHS) is a non-terminal and α (the RHS) is a possibly empty expression involving terminals, non-terminals, the disjunction operator (or alternative, $|$), the grouping operator (left and right parentheses), the Kleene closure (the $*$ operator, denoting the repetition of its argument 0 or more times), and the sequencing operator (concatenation of terms). One of the non-terminals is the *start* non-terminal and its corresponding production is the *main rule*.

In this paper, only regular grammars are considered. We thus restrict grammars to productions that do not lead to any form of recursivity, i.e. for each $A \in V$, it must not be possible to obtain $A \stackrel{*}{\Rightarrow} \alpha A \gamma$, for all α, γ.

Also, we restrict our grammars to have only one production for each non-terminal. This is not a limitation, but simplifies the description of the algorithms.

[1] Work done while Francis Picard and Dominique Boucher were at Infospace, 460 Ste-Catherine W # 801, Montréal, Canada H3B 1A7

[2] The work of Francis Picard was supported by a NSERC student grant.

A *word graph* is a graph where each vertex is labelled with a word of the input grammar. There are also two distinct vertices S and E representing the start and end of an utterance. A phrase accepted by a word graph is a path in the graph starting at S and ending at E.

1.2. Expansion

The first step in the conversion of a grammar into a word graph is called *expansion*. Its presentation is necessary to the description of our method. This step consists in substituting, in the main rule of the grammar, each occurrence of a non-terminal symbol by the RHS of its corresponding rule, and repeating this process until there is no more non-terminal in the main rule. At the end, we obtain a single rule, that we will call the *expanded grammar*. This expanded grammar can however contain a certain amount of redundancy, which our method will try to eliminate. We will not see the rest of the word graph construction method, (which is described in [6]), but it is important to note the exact correspondence between the expanded grammar and the final graph obtained. Note that to guaranty this correspondence, each occurrence of a terminal in the expanded grammar must correspond to a single vertex in the constructed graph and vice-versa.

2. METHOD

We will now describe a method for reducing the size of the resulting graph, while preserving the language recognized by the graph. Our method eliminates the redundancies at the grammar level, to avoid them in the final graph. We will call this task **grammar compression**.

The first technique is the left and right factorization of the different sequences of an alternative. Whenever we find, in a production, an alternative of the form:

$$\alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n \mid \gamma$$

where $|\alpha| > 0$, $n \geq 2$, and γ represents all the sequences of the alternative not beginning with α, we can **merge** the prefixes α to obtain:

$$\alpha(\beta_1 \mid \beta_2 \mid \cdots \mid \beta_n) \mid \gamma$$

Right factorization proceeds in a similar way.

This factorization process eliminates the redundancies between the different sequences of an alternative, but it cannot eliminate all the redundancies of the graphs. Consider the following grammar:

$$A \rightarrow a\, b \mid c\, (b \mid d)$$

It is impossible to have less than two occurrences of `b` in the grammar while the word graph representing the same language only needs one, as is shown in figure 1.

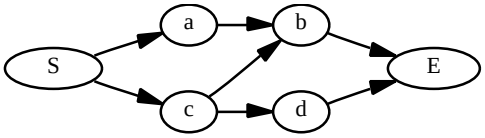

Fig. 1.

We can see that it is not always possible to have a bijective correspondence between the vertices of a word graph and the occurrences of the terminals in the expanded grammar.

To remedy this situation, we introduce the notion of sharing, in prefix or suffix. We mark two terms as shared to indicate to the construction method that they have to give rise to a single vertex or sub-graph in the final word graph. Thus, the two occurrences of b can be shared in suffix in the rule A. In general, the terms that can be shared in prefix between μ and ν, two sequences of a given alternative, are determined by $\Pi(\mu, \nu)$. This function returns the set of pairs of terms that can be shared in prefix. It is defined by the following formula:

$$\begin{aligned}
\Pi(\tau_1\gamma,\ \tau_2\varphi) &= \{(\tau_1,\ \tau_2)\} \cup \Pi(\gamma,\ \varphi) \\
&\quad \text{if } \tau_1 = \tau_2 \\
\Pi((\alpha_1|\cdots|\alpha_m)\gamma,\ (\beta_1|\cdots|\beta_n)\varphi) &= \bigcup_{\substack{1\leq i\leq m \\ 1\leq j\leq n}} \Pi(\alpha_i,\ \beta_j) \\
\Pi((\alpha_1|\cdots|\alpha_m)\gamma,\ \beta) &= \bigcup_{1\leq i\leq m} \Pi(\alpha_i,\ \beta) \\
\Pi(\alpha,\ (\beta_1|\cdots|\beta_n)\varphi) &= \bigcup_{1\leq j\leq n} \Pi(\alpha,\ \beta_j) \\
\Pi(\alpha,\ \beta) &= \emptyset
\end{aligned}$$

where τ_1 and τ_2 are either terminals, non-terminals, or parenthesized expressions. Note that for each pair of terms, one and only one of the above equations apply.

Sharing in suffix is symmetrically defined. Note that with $\mu = \alpha\gamma$ and $\nu = \alpha\varphi$, it is possible to factorize and sharing is not needed. In practice, sharing is done only when identicals terms cannot be factored because one is part of an alternative and not the other, or they are part of two different alternatives.

To compress a grammar or a rule, we simply scan it in search of redundancies amongst the sequences of its alternatives and we eliminate them with the factorization or the sharing, as is best suited.

However, it would be quite inefficient to expand the grammar completely and then apply our compression method. The expansion can sometimes lead to exponential growth in the size of the grammar. On the other hand, we could apply our technique on each rule of the grammar and then do its expansion. But we would not obtain as good a result

because there is often a lot of redundancies between the different rules of the same grammar, and these redundancies are multiplied by the expansion. Hence, we will apply the compression method incrementally during the expansion of the grammar. To do this, we have to do the expansion in several steps and apply the compression method to the main rule of the grammar after each step of expansion.

To expand the grammar incrementally, we first divide the set of non-terminals in a number of disjoint sets, and then do the expansion-compression of each set successively (that is, we substitute the references to the non-terminals in a given set with the right-hand side of their corresponding rules and then apply the compression algorithm). To form those sets, we use the partial ordering "refers to". We say that A "refers to" B iff there is a derivation $A \stackrel{*}{\Rightarrow} \alpha B \gamma$ from the grammar. A topological sort is then applied to the set of non-terminals according to this relation and successive incomparable elements are grouped together.

This algorithm deserves a few comments. First, it ensures that, once all references to a given non-terminal have been expanded in the main rule, the non-terminal can be removed from the grammar as well as its corresponding production rule. That is, no more references to that non-terminal appear in the grammar. Also, the expansion of a non-terminal is only performed once all references to that non-terminal belong to the main rule.

This algorithm therefore eliminates rendundancies in a top-down manner. If their elimination was further delayed (by using a more bottom-up approach), they would fatten up in the expansion process, and therefore the next expansion and compression steps would take more time.

In this process, we have to be careful not to share a term in prefix as well as in suffix, because this generally modifies the language recognized by the grammar. Also, the factorization is not essential, because we could eliminate all the redundancies with sharing only. We kept using factorization because it has the advantage of reducing the representation of the grammar. Morever, the expansion and compression steps are faster when using factorization.

3. EXPERIMENTAL RESULTS

We have implemented the expansion-compression method in the InfoSpace dynamic grammar compilation server. To evaluate its effectiveness, we compared it with the classical automaton determinization and minimization algorithms. In the latter case, the word graph is obtained by (1) expanding the grammar completely, (2) converting it into a word graph, (3) converting the word graph into an automaton, (4) determinizing and minimizing the automaton, and (5) converting the resulting automaton back to a word graph. The determinization and minimization algorithms have been taken from the AT&T FSM library[7]. All tests have been conducted on a Sun Ultra-10 336 MHz with 512 megabytes of RAM running Solaris Sparc.

Grammar	*Exp.*	*Minim.*		*Exp.-Comp.*	
	size	*time*	*size*	*time*	*size.*
DurationTime	1165	0.08	**237**	**0.06**	328
SocialInsNo	1680	0.70	447	**0.37**	**381**
DigitQuad	1810	**0.44**	275	0.73	**244**
FromToTime	2504	0.26	474	**0.07**	**462**
Banking	2721	0.55	**260**	**0.05**	278
BirthDate	2806	0.78	281	**0.34**	**255**
DollarAmountSmall	3017	2.35	339	**0.71**	**338**
FromToMoNaYear	3108	**0.32**	532	0.63	**530**
DigitQuintFr	3324	**0.57**	427	1.03	**355**
DigitQuint	3490	**0.59**	579	1.72	**458**
FromToDate	3628	1.00	658	**0.65**	**656**
FromToMoNaYearFr	5191	0.52	399	0.52	**397**
NoMaxNineDigits	5293	**0.26**	245	0.72	**243**
DollarAmountBig	6035	4.45	450	**0.74**	**449**
CreditCardNumber	7240	13.42	2147	**2.40**	**976**
DollarAmountSmallFr	7586	4.71	866	**1.34**	866
SpeechAct	9382	5.06	669	**0.09**	**470**
AtisSimplified*	96745	27.78	25268	**2.42**	**24274**
Atis*	148905	N/A	27542	**13.90**	**26509**

Table 1. Comparison between exact minimization and our method.

Table 1 gives the size of the resulting word graphs and the time (in seconds) taken by the two compression methods on 19 grammars. The first column gives the grammar name. Column 2 gives the size (number of vertices) of the word graph obtained using the expansion algorithm only. Columns 3 and 4 give the time taken by the minimization algorithm only and the size of the resulting word graph. Finally, columns 5 and 6 give the time taken by our expansion-compression method as well as the size of the resulting word graph. The asterisks indicate that the corresponding grammar has some terminals that are references to some imported grammars. These references are replaced by their corresponding language model only in the recognition engine at loading time. It is the resulting number of words in the recognition engine that is shown here. The number of words in the importing grammar is thus smaller.

Note that it was not possible to apply the minimization algorithm on the word graph for the Atis grammar without compression (the input graph has 74985 vertices) since it required too much memory. (The number of words has been obtained by minimizing the word graph obtained with our method). We thus simplified the grammar a bit to obtain the AtisSimplified grammar.

Table 2 gives the speech recognition time for three grammars on a single utterance. Column 2 gives the average time obtained using the grammar without any compression,

Grammar	*Exp.*		*Exp.-Comp.*	
	time	*%wc*	*time*	*%wc*
DollarAmountSmallFr	1.83	84.67	0.49	84.67
DigitQuintFr	0.57	75.86	0.19	75.87
FromToMonthNameYearFr	1.61	78.16	0.75	78.16

Table 2. Mean recognition time in seconds (*t.*) and word percentage correct (*%wc*)

while column 4 gives the average time using the grammar compressed using our method. Recognition accuracy (% of words correct) is also shown, but it is barely identical in both cases. We observed that the recognition grammars obtained using our method require less memory to load in the speech recognition server. For example, the memory required to load the grammar Atis went down from 204 to 30 megabytes. For smaller grammars, reduction in memory usage was not as impressive, but it was still noticeable.

4. DISCUSSION

In most cases, our method gives rise to word graphs smaller than those obtained using exact minimization. This is because it avoids the determinization step, which can lead to exponential growth. The CreditCardNumber grammar greatly benefits from our method. But in most cases, the differences are not as dramatic.

In some other cases, however, our method does not perform as well as exact minimization. It is by nature a heuristic method and there are some redundancies that it fails to eliminate. For example, the fact that it prevents the sharing of a term in prefix and suffix at the same time can sometimes block some potential compression. There are some others cases where our method lets some redundancy pass (see [6] for details). They are all very specific cases that it would have been possible to eliminate, but at a prohibitive cost.

The processing times of table 1 show that, for small grammars, our method performs better than exact minimization. For bigger grammars, the advantage of our method becomes clearer. This is mainly because it is more efficient to compress the grammar periodically than building the whole graph and compressing afterwards.

We also observe that processing time for our method varies greatly from one grammar to another. This is mainly due to the fact that disjunctions with a lot of alternatives will incur a bigger compile time performance penalty.

One of the advantages that our method offers against exact minimization is its greater flexibility. For example, it has been possible to limit the search for redundancies to the parts of the main rule that were modified in the previous expansion step. That would not have been possible with a classic minimization algorithm. It enabled us to obatin significant improvements in processing time. It would also be possible to control the trade-off between time of processing and reduction in size by applying or not the compression algorithm after the last expansions, when it takes the most time (see [6] for more details on those two points).

The fact that our method is a heuristic has a drawback. It is not guaranteed to perform well with all grammars. However, in practice, our method almost always achieves a reduction in size as good or better than with exact minimization. And it more often does significantly better than significantly worse. It avoids exponentially growth in all but degenerated cases. For a more thorough description of the method, see [6].

5. CONCLUSION

This paper have presented a heuristic method for constructing a word graph from a regular grammar that leads to language models comparable in size with ones obtained using classical automaton determinization and minimization algorithms. The main advantage of our method is that it requires less resources (both in processing time and memory usage) than exact minimization and is much more flexible.

6. REFERENCES

[1] P. Kenny, R. Hollan, V. N. Gupta, M. Lenning, P. Mermelstein, and D. O'Shaughnessy, "A*-admissible heuristics for rapid lexical access," *IEEE Trans. Speech and Audio Proc.*, vol. 1, pp. 49–57, 1993.

[2] R. Lacouture and R. D. Mori, "Lexical tree compression," *In Proc. 2nd EUROSPEECH*, vol. 2, pp. 581–584, 1991.

[3] M.K. Brown and J.G. Wilpon, "A grammar compiler for connected speech recognition," *IEEE Trans. on Signal Proc.*, vol. 39, no. 1, pp. 17–38, 1991.

[4] A.J. Buchsbaum, R. Giancarlo, and J.R. Westbrook, "Shrinking language models by robust approximation," *Proc. of the 1998 IEEE ICASSP*, vol. 2, pp. 685–688, 1998.

[5] M. Mohri, "Finite-state transducers in language and speech processing," *Comp. Ling.*, vol. 23, no. 2, pp. 269–311, 1997.

[6] Francis Picard, "Génération de modèles de langage compacts pour la reconnaissance vocale," M.S. thesis, Université de Montréal, 2002, To be completed.

[7] M. Mohri, F.C.N. Pereira, and M.D. Riley, "Fsm library," Available at http://www.research.att.com/sw/tools/fsm/.