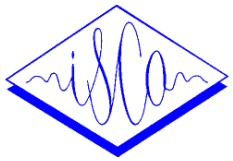
ISCA Archive
http://www.isca-speech.org/archive



International Symposium on Chinese Spoken Language Processing (ISCSLP 2002)
Taipei, Taiwan
August 23-24, 2002

EQUIVALENT NODE-BASED SPEECH GRAMMAR OPTIMIZATION

Min ZHANG *Cuntai GUAN* *Haizhou LI*

Infotalk Technology, Republic of Singapore
{min.zhang, cuntai.guan, haizhou.li}@infotalkcorp.com

ABSTRACT

In a domain-specific speech application, for example in a computer-telephony dialogue system, a speech grammar is used to specify the words and patterns of words to be understood by a speech recognizer. The size of the recognizer search space depends on the speech grammar perplexity. In this paper, we propose a novel equivalent node-based speech grammar optimization algorithm. The proposed algorithm can optimize the grammar-based search graph by iteratively combining the equivalent nodes and their fan-in & fan-out transitions (arcs) if local properties show that this does not change the grammar coverage. Experiments on a 70k Resident Identity Numbers show that the proposed method can reduce 80% nodes and transitions, and the recognition is also found to be 50% faster without affecting accuracy. The algorithm is proven to be high effective and efficient.

1. INTRODUCTION

A speech grammar is used to define the words and the patterns of words to be understood by a speech recognizer in domain-specific applications, ranging from voice-command-control to computer-telephony dialogue systems. Speech grammar can constrain the user's words, which is one of the most popular configurations for building limited-domain speech applications because of the following advantages [1]:

- Compact representation that results in a small memory footprint.
- Efficient operation during decoding in terms of both space and time.
- Ease of grammar creation and modification for new tasks.

In general, a speech grammar should be compiled into recursive transition network (RTN) or further into finite state automata (FSA). HMMs are embedded into RTN or FSA, based on which speech engine can do recognition.

We define the grammar perplexity as the total sum of the number of the grammar nodes and transitions. Grammar perplexity is a good indicator of search complexity of a recognizer. The higher the grammar perplexity is, the greater the search complexity is.

In this paper, we propose a novel equivalent node-based speech grammar optimization algorithm. The proposed algorithm can optimize the grammar-based search graph by iteratively combining the equivalent nodes and their fan-in & fan-out transitions if local properties show that this does not change the grammar coverage.

Experiments on a 70k Resident Identity Numbers show that the proposed method can reduce 80% nodes and transitions, and the recognition is also found to be 50% faster without affecting accuracy as compared to no optimization test. Furthermore, in some other testing cases, the proposed algorithm not only reduces memory and speeds up recognition, but also improves the accuracy due to more effective search.

Another advantage of the new algorithm is to allow for online grammar optimization with small memory requirement.

The paper is organized as follows. In Section 2, we review the different specifications of speech grammars and another commonly used optimization algorithms. In Section 3, we discuss the proposed equivalent node-based grammar optimization algorithm. In Section 4, we give the experimental results. In Section 5, we discuss the semantic representation and semantic action in speech grammar and their influences on the optimization.

2. GRAMMAR AND ITS OPTIMIZATION

Up to now, a lot of speech grammar formats have been defined and applied to different tasks, such as Microsoft SAPI grammar [2], Sun Microsystems Java Speech grammar [3], W3C ABNF and XML grammars [4], Infotalk GDL grammar [5], Nuance GSL grammar [6] and Speechworks grammar [7].

Regardless of specifications described above, in general, the speech grammar is clearly characterized by traditions of formal language theory. Regardless of the semantic actions in grammar and some constrains in practice, the speech grammar has the same description capability as context-free grammar (CFG).

In general, a speech grammar consists of a set of productions or rules, which expand non-terminals into a sequence of terminals and non-terminals. One non-terminal is designated as the start rule of the grammar. Speech grammar also allows some regular expression operators, such as “or”, “repeat” and “optional” operations.

Without losing generality, speech grammar can be formulated with recursive transition network (RTN). Different from FSA, RTN allows transition label to refer to other networks as well as terminals. RTN can be further expanded to FSA by replacing the reference transition with the real referred network[1]. Due to the speed consideration, our system employs the FSA embedded with HMMs at transition label as the search space (or search net).

The grammar perplexity is defined as the total sum of the number of nodes and transitions of the grammar internal representation --- RTN or FSA. For a small grammar, the perplexity is very lower. But for a so-called "big" grammar, for example a grammar used in Yellow Pages system for telephone directory service, the node and transition number can be over one million. This results in a significant increase of search effort and recognition time. To maintain a reasonable responding time, a common approach in this case is to adjust pruning threshold, often leading to accuracy drops.

Grammar optimization algorithm can help resolve the problem by reducing the search space. A grammar optimization algorithm will try to reduce the number of nodes and transitions, and at the same time, should not reduce the grammar coverage that can be defined by the total number of sentence paths this grammar could generate. From the formal language viewpoint, the optimized grammar should have the exactly identical expression capability as the un-optimized grammar. Therefore, a proper grammar optimization algorithm can both improve the search speed and reduce the memory requirement. It could even reduce word error rate in some cases, for example a grammar with a large number of branches. This is due to the fact that a practical real-time recognizer always adopts pruning schemes.

3. EQUIVALENT NODE-BASED GRAMMAR OPTIMIZATION

The classical FSA minimization algorithm [12] is very computationally expensive, with the complexity of O(n^2). In addition, FSA minimization algorithm only works on the deterministic FSA and FSA determinization requires exponential worst-case time and space complexity [8][12]. We thus chose to develop a fast optimization algorithm to work directly on non-deterministic FSA by clustering the equivalent nodes to eliminate local redundancies. The key concept of the algorithm is equivalent node.

3.1 Equivalent node

Equivalent node definition: If two nodes in FSA have the same set of successor (or predecessor) transitions, and the two nodes have the same father node, then they are called equivalent node. Depending on successor or predecessor transitions set, the equivalent node can be classified as "forward" or "backward" equivalent node.

[1] In theory, only regular grammar can be represented by FSA, in other words, not all of the RTN can be expanded into FSA exactly. Thus, several algorithms [9][10] have been proposed to use FSA to approximate RTN. This simulation will bring bias in the coverage between RTN and FSA; FSA will be of over-generation than RTN. In our system, the result-understanding module can repair the bias, please see section 5.

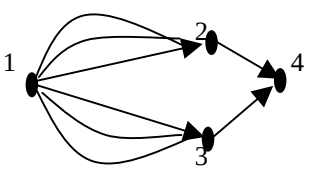


Figure 1. FSA (1)

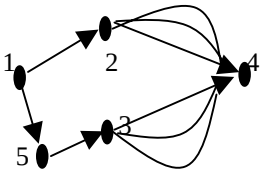


Figure 2. FSA (2)

In Figure 1, Node 2 and Node 3 are "forward" equivalent node. They have the same father node 1, and we assume that the three fan-in transitions of Node 2 are identical with those of Node3. In Figure 2, Node 2 and Node 3 are "backward" equivalent node. They have the same father Node 4, and we assume that the three fan-out transitions of Node 2 are identical with those of Node3.

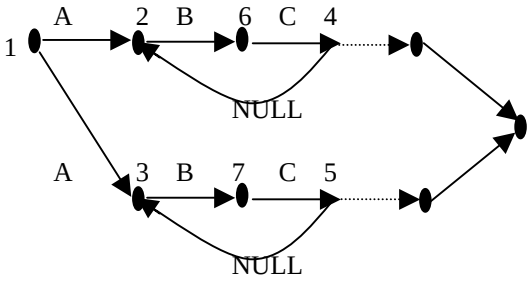


Figure 3. FSA (3)

Node 2 and Node 3 in Figure 3 are another example of "forward" equivalent node, but this case is not so straightforward, because it depends on whether Node 4 and Node 5 are "forward" equivalent node. This case only happens when the grammar supports "repeat" operator. Node 2 and Node 3 are called **key-repeatable node**, which will be labeled with a flag for the special processing in optimization.

Obviously, the equivalent node can be merged to reduce the redundant search space without changing grammar coverage. In the next section, the optimization algorithm is discussed in detail.

3.2 Optimization Algorithm

In our system, the algorithm is applied in three levels respectively:

- RTN level --- transition with terminal/non-terminal label.
- Expanded RTN level --- transition with terminal label.
- Search Space Level ---- transition with HMM label.

In each level, the algorithm scans grammar twice by either depth-first or breadth-first search. The first is the forward pass, which scans the grammar from the beginning node to ending node and merges the forward equivalent nodes. The second is the backward pass, which scans the grammar from the ending node to the beginning node and merges the backward equivalent nodes. For brevity, we describe only the forward optimization algorithm at expanded RTN level; others are symmetrical.

Forward optimization algorithm: Let $S_{out\_repeat}(n)$ be the set of successor nodes of Node *n* and all of nodes in this set must be **key-repeatable nodes;** let $S_{out\_other}(n)$ be the set of other successor nodes of Node *n*.

foreach (node ***n*** in FSA) // *depth-first or breadth-first*

1. Delete identical transitions[1].
2. Construct two sets: $S_{out\_other}(n)$ and $S_{out\_repeat}(n)$.
3. Partition $S_{out\_other}(n)$ into sub-sets, all nodes in which are equivalent.
4. Partition $S_{out\_repeat}(n)$ into sub-sets, all nodes in which are equivalent.
5. Merge the equivalent nodes in each sub-set[2].

Partition $S_{out\_other}(n)$**:** Let $A_{arc}(n)$ be an array of all of fan-in transitions of the nodes in the set $S_{out\_other}(n)$.

1. Sort $A_{arc}(n)$ according to the destination node and label of the current transition.
2. Partition $A_{arc}(n)$ into a series of segments, and transitions in each of which has the same destination node.
3. Sort these segments by transition number and comparing transition label one by one if transition numbers are identical.
4. Traverse the series of segments, compare each segment with its previous segment, if the two segments are identical[3], then destination node of the transitions in the two segments are equivalent.

Partition $S_{out\_repeat}(n)$ **:** similar to the function **Partition** $S_{out\_other}(n)$.

1. Assume the nodes in the set $S_{out\_repeat}(n)$ are equivalent.
2. Iteratively call the function **Partition** $S_{out\_other}(n)$ by either depth-first or breadth-first search.
3. If **Partition** $S_{out\_other}(n)$ returns false in any step during the iterative processing, it means that the two nodes are not equivalent; otherwise they are equivalent nodes.

[1] Two transitions are identical if their labels, source and destination nodes are identical.

[2] When merging the equivalent nodes, only one copy of the fan-in transitions needs to be kept, but all of the fan-out transitions should be merged.

[3] Two segments are identical if they contain the same transitions set.

Each node and transition in the FSA are only visited once in the optimization processing, even if in the function **Partition** $S_{out\_repeat}(n)$, so the runtime for this algorithm is O(n).

The key idea behind this algorithm is to define the equivalent node and transition-sorting-based equivalent node partition algorithm. The optimization algorithm classifies all the nodes in grammar into some equivalent node classes by transition sorting partition algorithm. Once the equivalent nodes have been clustered, they can be merged easily to optimize the grammar.

4. EXPERIMENTAL RESULTS

The proposed optimization algorithm is very effective for RTN or FSA-based speech grammars, which are widely used in the state-of-the-art telephony speech dialogue system.

We select a sequence of grammars to carry out testing. The grammar entry numbers are from 1k to 70k. The grammars grammar with 1k, 10k and 40k entries are for Chinese Yellow Pages. The grammar with 70k entries is for English Resident Identity Numbers.

Table 1 illustrates the testing results. The testing results show consistent improvements. For example, with the grammar composed of 70k Resident Identity Numbers, our approach reduces 88.3% nodes and 79.7% transitions, that is to say, about 80% redundant search space is reduced by the algorithm. Meanwhile, recognition speeds up by 52% after optimization without affecting recognition accuracy. For small grammars, the algorithm still works well on reducing memory requirement, but has less impact on speed and accuracy because the search space does not reach the ceiling of the pruning beam.

Furthermore, in testing cases of the grammars with 70k entries, the proposed algorithm not only reduces memory and speeds up recognition, but also improves the accuracy. It is due to the fact that a practical real-time recognizer always adopts pruning schemes.

The speed of the optimization algorithm is very quick. Table 1 lists the speed of the algorithm (IBM PC P4 1GHZ). The speed allows the online grammar optimization.

In conclusion, the optimized grammar leads to a more effective search.

5. DISCUSSION

We have given the definition of equivalent node and presented the equivalent node-based grammar optimization algorithm. Experimental results reveal that the proposed algorithm is very effective and efficient for big grammar. The algorithm can reduce the search space and speed up the recognition time without sacrificing the recognition accuracy.

Speech grammar has twofold functions in a speech dialogue system: one is to construct the search space for recognizer; another one is to get the semantics out from the raw recognition result in dialogue system (speech in, semantics out). Although the proposed algorithm cannot work on the semantic grammar directly, but there is a solution to it. In our system, a RTN for semantic parsing is built from the text grammar firstly, from

which, a RTN counterpart for expanding is built by eliminating the semantics-related information. Then we can apply the algorithm on the RTN counterpart. In short, our algorithm only works on the RTN without semantics information, and the RTN with semantics information is only for semantic parsing.

Even though we can just apply the algorithm in the HMMs level FSA, due to consideration of efficiency, we still apply the algorithm in three levels, this is because it is more efficient to optimize RTN directly compared with the FSA expanded from the same RTN.

Our current algorithm is a lossless optimization algorithm; that is to say, the algorithm does not change the grammar coverage. A more aggressive algorithm is to increase the grammar coverage during the optimization processing by relaxing the condition of equivalent nodes[1]. This can further reduce the search space; in the meanwhile, our semantic parsing can guarantee that only the results covered by original grammar will be outputted. We have not yet evaluated the tradeoffs associated with the optimization with loss.

6. REFERENCES

[1] XueDong Huang, Alex Acero and Hiao-Wuen Hon, "Spoken Language Processing", pressed by Pretice-Hall Inc. 2001.

[2] Java Speech Grammar Format, http://www.w3.org/TR/jsgf/

[3] Microsoft SAPI grammar, http://www.microsoft.com/speech

[4] Speech Recognition Grammar Specification for the W3C Speech Interface Framework, http://www.w3.org/TR/2002/CR-speech-grammar-20020626/

[5] Infotalk GDL Speech Grammar, www.infotalkcorp.com

[6] Nuance GSL Speech Grammar, www.nuance.com

[7] Speechworks Speech Grammar, www.speechworks.com

[8] F.Weng, A.Stolcke, and A.Sankar. "Efficient Lattice Representation and Generation". *In the Proceeding of International Conference on Spoken Language Processing, Sydney*, Australia, Nov.30 – Dec.4, 1988

[9] Fernando C.C. Pereira and Rebecca N. Wright, Chapter 5 in "Finite-State Language Processing", edited by Emmanuel Roche and Yves Schabes, pressed by MIT Press, 1997

[10] Mark-Jan Nederhof, "Practical Experiments with Regular Approximation of Context-Free Language", Computational Linguistics, Vol 21, No. 1, 2000, pp.17~44

[11] Horacio Franco, Jing Zheng et al. "Dynaspeak: SRI's scalable speech recognizer for embedded and mobile systems", International Conference of Human language Technology 2002, San Diego, CA, 2002, pp 23-26.

[12] J.E.Hopcroft and J.D.UIIman. "Introduction to Automata Theory, Language, and Computation", Addison-Wesley, Reading, MA, 1979

Grammar Entry Number	1k	10k	40k	70k
Node number (before optimization)	6375	58082	231651	761108
Node number (after optimization)	1981	15880	60996	89002
Reduction rate	68.9%	72.6%	73.6%	88.3%
Transition number (before optimization)	5347	48136	191076	761106
Transition number (After optimization)	3038	25843	100733	154005
Reduction rate	43.1%	46.3%	47.2%	79.7%
Speed	0.60s	1.29s	4.24s	12.74s

Table 1. Grammar Optimization Testing Results

[1] This can be done by changing the fourth step in the "partition" function from the exactly equal to a certain percentage of identical transitions.