Efficient Word-graph Parsing and Search with a Stochastic Context-free Grammar

C.J. Waters **B.A. MacDonald**
Department of Electrical and Electronic Engineering
The University of Auckland
Auckland, New Zealand

Abstract - Word-graphs provide a compact representation for the output of the acoustic decoder in a speech recogniser. This paper proposes an efficient algorithm for parsing the graph structure using a stochastic context-free grammar (SCFG). By parsing an entire graph in a single pass significant savings can be made over techniques that parse individual sentences, such as the common N-best strategy, or methods that parse paths through a graph. A backward Viterbi search is used to recover the parsed sentences from the graph. The full graph parsing algorithm is shown to be better than a heuristic search that parses only portions of the graph.

On the Resource Management task a reduction in computation of 5 times over N-best is demonstrated. The integration a SCFG gives a reduction in recogniser word error rate of 9.8%.

1 Introduction

Word-graphs [5] are a popular mechanism for representing sentence hypotheses between different stages of a speech recogniser. They provide a compact representation of spoken words that is suitable for efficient search. This has led to their widespread use in many large speech systems [4, 1].

In [10] we present an algorithm for efficiently parsing an entire word-graph using a stochastic context-free Earley parser [6]. In this paper we evaluate an alternative heuristic search method that parses only promising sections of the word graph, and present empirical results showing that the complete graph search is more effective than the alternative, heuristic method.

The entire word-graph parsing technique is able to parse word-graphs containing millions of sentence hypotheses in a few seconds. Parsing the entire word-graph would appear wasteful when most of the time we are interested in the single best grammatical sentence or at most a few hundred sentences. An alternative method would be to use a search technique such as A^* combined with the acoustic and bigram scores to direct the parser to parse only portions of the graph that are acoustically most likely. If the best acoustic

0-7803-3698-4/97/$10.00 © 1997 IEEE.

path[1] through the graph is also grammatical then only that path is parsed making the technique very efficient. If however many of the paths that score well acoustically are not grammatically correct or in fact no grammatical path exists then large portions of the graph will be parsed.

The aim of the investigation was to compare the empirical performance of the full graph parse and A^* methods as opposed to the baseline technique of extracting N-best sentences and parsing them one by one.

2 Word-graphs

The word-graphs we use are generated by ARISTOTLE [9], a HMM based speaker-independent recogniser, on the Resource Management corpus. Words are attached to edges in the graph. Since the graphs are generated online by the acoustic processor they also contain information about silence and garbage words. Before the graph can be parsed all silence edges are removed from the graph with a single traversal. It may be fruitful to consider using a technique similar to those used for removing nondeterminism from finite state automata [7] for silence edge removal. This would also tend to reduce graph density by removing duplicate paths.

3 Heuristic Word-graph Searching

An heuristic, A^*, search chooses the best path to follow based on the cost to the current edge and a heuristic cost to the end of the path. Thus we need an estimate for the lower bound on the cost from the current edge to the end of the sentence. A fast backward pass over the graph computes the backward Viterbi cumulative score which becomes the lower bound estimate. This estimate is admissible since we are using the product of P_{bigram} and P_{SCFG} as our language model which will always result in a value that is less than or equal to the bigram probability alone. If the word-graph is created during a traceback phase then cummulative score can also be computed at that stage removing the need for an extra graph traversal.

To extract the best sentence from the graph we perform a forward pass over the graph using the Earley graph parsing algorithm with A^* choosing the sentences to expand at each parsing step. The A^* search ensures that only likely paths are considered. The graph is expanded into a tree as the search proceeds and so paths are not recombined when they rejoin in the graph.

4 Earley Parsing

The top down Earley parsing algorithm [3] operates in $O(n^3)$ time on any context-free grammar (CFG). It was chosen over the generalised LR method of

[1] The words 'path' and 'sentence' are used interchangably—by definition a complete path through the graph corresponds to a sentence.

Tomita [8] because of the existence of a general, efficient and exact method for computing probabilities when using stochastic CFGs with the Earley method [6]. Also the performance of Earley's method is improved when precomputed matrices of parsing hints are created.

In this section we use the following notation. The input string, x, to be parsed has length $|x|$ and is made up of terminal symbols $x_0, x_1, ..., x_{|x|-1}$ from some finite alphabet Σ, the elements of which are denoted by lowercase roman letters $a, b, c, ...$. Nonterminal symbols are identified with uppercase roman letters X, Y, Z etc. The parser manipulates arbitrary strings λ, μ, υ of terminal and nonterminal symbols. The empty string is denoted by ϵ.

The Earley parser operates on sets of *Earley states* of the form:

$$i : {}_kX \rightarrow \lambda.\mu \tag{1}$$

where i is the current position in the input and also the state set identifier. $X \rightarrow \lambda\mu$ is a production from the grammar that was first hypothesized at position k in the input. The dot shows the current position in the parse.

The parser moves the dot through productions in the grammar until the entire input string has been processed. A state with the dot on the extreme right of the right-hand side of the rule is a *complete state.* The parser uses three operations to create new states and add them to the state sets.

The three operations are:

Prediction For each state of the form:

$$i : {}_kX \rightarrow \lambda.Y\mu$$

and all rules $Y \rightarrow \gamma$ add the state:

$$i : {}_iY \rightarrow .\gamma$$

This operation expands eligible grammar rules creating new *predicted states.*

Scanning For each state of the form:

$$i : {}_kX \rightarrow \lambda.a\mu$$

and the next input token $x_i = a$ add the state:

$$i+1 : {}_kX \rightarrow \lambda a.\mu$$

This operation consumes input and creates *scanned states.* Scanning is the only operation that creates states in a different state set, $i+1$, from the motivating state, i. Essentially it moves the dot over a terminal symbol.

Completion For each complete state:

$$i : {}_jY \rightarrow \gamma.$$

and each state in state set j that has Y to the right of the dot:

$$j : {}_kX \rightarrow \lambda.Y\mu$$

add the state:

$$i : {}_kX \rightarrow \lambda Y.\mu$$

The completion operation corresponds to identifying a substring in the input that has been correctly parsed. The new state created is called a *completed state.*

The parser is seeded by creating a state in the first state set with the form

$$0 : {}_0 \rightarrow .S \tag{2}$$

where S is the left hand side of the designated initial rule of the grammar. Note the empty left-hand side of the rule being used here is not from the grammar but is a psuedo rule used to intialise the parser.

The three operations are applied repeatedly and exhaustively to all of the state sets until no new states are created. When the parser tries to create a new state for which an identical state already exists then a duplicate state is not created. Because of this it should be clear that only a finite number of states could be created. There are only a finite number of grammar rules, positions in the input and positions for the dot so the process must terminate.

If in the final state set, $|x|$ there is a state of the form

$$|x| : {}_0 \rightarrow S. \tag{3}$$

then a legitimate parse of the input exists.

5 Parsing a Word-graph

The Earley parsing algorithm can be adapted to parse all of the sentences contained in a word-graph simultaneously. In the standard algorithm state sets are numbered for the position in the input they correspond with. To maintain the relationship between words and state sets when extending to word-graph parsing a state set is attached to each edge, since in our graphs words are on edges. If the edges of a graph are numbered uniquely in some arbitrary way then the same numbering scheme can be used for the state sets.

The scanning operation is also modified since it can create states in a different state set. In the standard algorithm the new scanned state is created in the state set of the next word. In the extension to word-graph parsing the state set of every edge $e_{j_1}, e_{j_2}, ...$ that can directly follow the current edge e_i needs to be populated with an identical scanned state. This ensures a correct parse for all sentences passing through the branches $(e_i, e_{j_1}), (e_i, ej_2), ...$. Prediction and completion are done as before.

The algorithm is best illustrated with an example. The simple grammar:

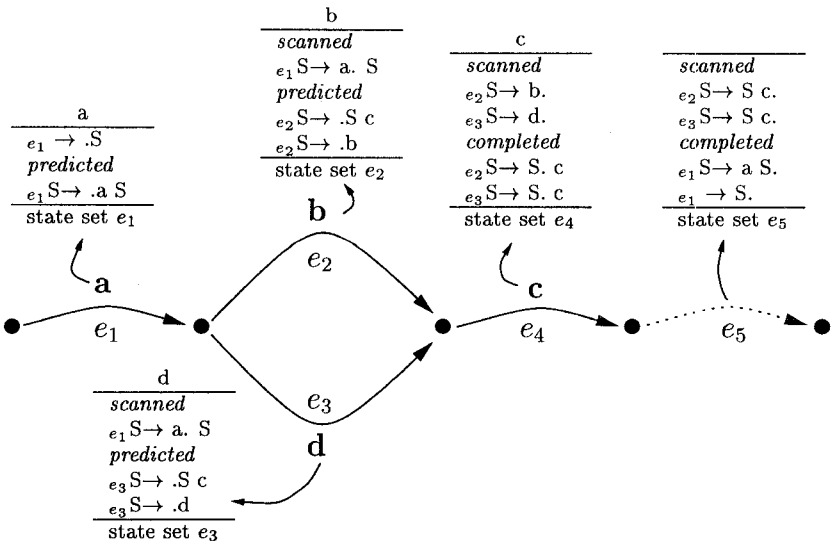

Figure 1: State sets created during graph parsing.

$$\begin{array}{lcl} S & \rightarrow & a\ S \\ S & \rightarrow & S\ c \\ S & \rightarrow & b \\ S & \rightarrow & d \end{array}$$

generates strings such as abc and adc. These two strings can be represented in a graph. Each edge of the graph is labeled, these labels become the state set identifiers. A dummy edge is created from the last vertex to hold the final state set.

Figure 1 shows the same graph after the parse is complete. The final state set, on edge e_5 contains the state e_1: $_{e_5} \rightarrow$ .S signifying that at least one correct parse was found in the graph.

Note how scanning on the state e_1: $_{e_1}$S $\rightarrow$ .a S creates two new states, one in each of the state sets e_2 and e_3. By state set e_5 no extra computation needs to be performed for the e_2, e_3 branches because the state e_5: $_{e_1}$S $\rightarrow$ a S. is generated for both. Once branches in a sub-graph have converged no extra information needs to be carried over for computation of the containing graph. This fact is at the core of the algorithm's efficiency.

As long as paths through the graph tend to recombine, and in actual word-graphs they do, then parsing the entire graph is far more economical that parsing an N-best list created from the graph.

6 Extracting Parse Results

As the graph is parsed a forest of parse trees is being constructed in the form of the parser states. There is one tree for each parsed path (multiple if

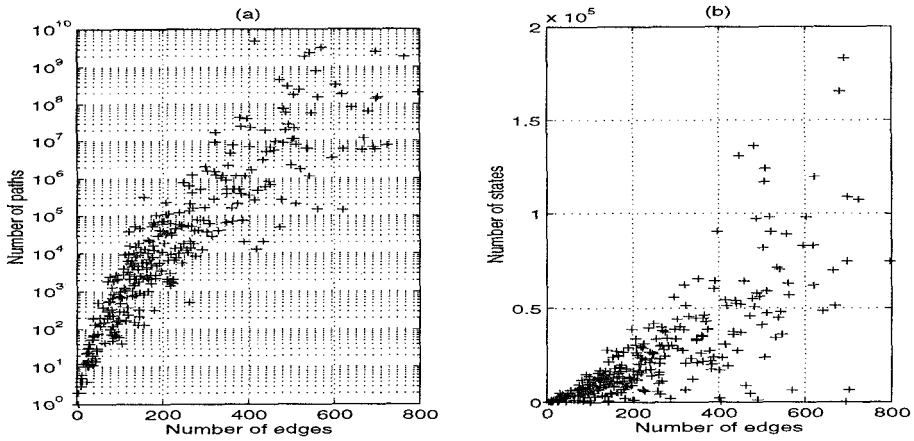

Figure 2: (a) The number of paths contained in each graph as a function of the number of edges. (b) Number of parser states generated for each graph plotted against the number of graph edges. Notice that the increase in parser states is roughly linear with an increase in edges.

the sentence is ambiguous) and thus a huge number of trees given the large number of potential paths. Because of the graph structure however the trees are factored together so that identical portions are shared.

The graph is rescored with the prefix scores computed by the SCFG. We incorporate both the bigram grammar and SCFG probablities as a product to maintain the benefit of both types of model. This technique forces the input to conform to the grammar while also lying amongst the best Markovian candidates [2]. When dealing with new input that may not be covered by the SCFG we can also apply a minimum threshold to P_{SCFG} to prevent ungrammatical but highly scoring hypotheses from being rejected.

Then using a backwards Viterbi search the best (and second best etc) sentence can be extracted.

7 Results

Both algorithms were tested on 330 sentences, 10 each from 33 speakers, from the Resource Management corpus August evaluation set. The word graphs used contained an average of 28 edges per word. Table 1 shows the results of the tests. For comparison, when the N-best sentences were extracted from the word-graphs and parsed an average of 260 Earley states per sentence were generated. Parsing each entire graph resulted in an average of 91 states per edge or 25,000 states per graph. The computation required to

Num. Parser States	N-best	Heuristic Search	Full Graph Parse
per sentence	260	49,900	0.00029
per edge/word	30.8	30.8	90.6
per graph	2×10^{10}	49,900	25,000

Table 1: System comparison.

extracted the parsed sentences from the graph is identical to extracting the N-best sentences. On average parsing an entire graph equates to the same amount of work as parsing the first 96 hypotheses in an N-best list. Previous experiments[10] have shown that 500-750 hypotheses are required in order to guarantee inclusion of the correct hypothesis and so the graph parsing technique is at least 5 times more efficient that parsing an N-best list. There was much more variability in the results from the heuristic search. In cases where the best acoustic path through the graph was also grammatically correct then only the minimum number of parser states were generated. When there was no grammatical path in the graph the entire search tree must be constructed and so a huge number of parser states were created, in some cases exhausting available memory resources before completing. Since the graph is being parsed as a tree the computation quickly becomes exponential in the number of edges in the graph. On average the heuristic search takes twice as long as a full graph parse and only generates the top hypothesis.

8 Summary

We have presented two algorithms for parsing sentences contained in a word-graph with a SCFG. The complete word-graph parsing algorithm gives better and more consistent results than the heurisc A^* search technique. Benefits of the word-graph parser are:

- parsing is done *in situ* without requiring an N-best list to be generated or sentence strings to be extracted.
- because of the compactness of the graph structure there is a huge reduction in parsing effort over N-best list or A^* techniques which must process the same substrings repeatedly or use a complicated memoization technique.
- the algorithm is linear in the number of graph edges and doesn't explode if the graph contains no grammatical sentences.
- multiple hypotheses can be produced for rescoring by another knowledge source. These could be extracted in the form of an N-best list or left in the graph for another graph-processing technique.

Acknowledgments

This work was carried out with the assistance of Telecom Corporation of New Zealand Ltd and the Centre for High Performance Computing at the University of Auckland.

References

[1] Xavier Aubert and Hermann Ney. Large vocabulary continuous speech recognition using word graphs. In *International Conference on Acoustics, Speech and Signal Processing*, pages 49–52, 1995.

[2] Anne-Marie Derouault and Bernard Merialdo. Natural language modeling for phoneme-to-text transcription. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):742–748, November 1986.

[3] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970.

[4] Hy Murveit, John Butzberger, Vassilios Digalakis, and Mitch Weintraub. Large-vocabulary dictation using sri's decipher speech recognition system: Progressive search techniques. In *International Conference on Acoustics, Speech and Signal Processing*, pages 319–322, 1993.

[5] M. Oerder and H. Ney. Word graphs: An efficient interface between continuous-speech recognition and language understanding. In *International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 49–52, 1993.

[6] Andreas Stolcke. An efficient probabalistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

[7] Thomas A. Sudkamp. *Languages and Machines.* Addison–Wesley, 1988.

[8] M Tomita. *Efficient Parsing for Natural Language.* Kluwer Academic Publishers, 1986.

[9] Christopher Waters and Bruce MacDonald. The ARISTOTLE speech recognition system. In *The Fifth Australian and New Zealand International Conference on Intelligent Information Processing Systems*, pages 1061–1064, 1997.

[10] Christopher Waters and Bruce MacDonald. Efficient parsing of word-graphs for speech recognition. In *The Fifth Australian and New Zealand International Conference on Intelligent Information Processing Systems*, pages 1084–1087, 1997.