NEAR MINIMAL WEIGHTED WORD GRAPHS FOR POST-PROCESSING SPEECH

Michael T. Johnson and Mary P. Harper

Purdue University, School of Electrical and Computer Engineering
West Lafayette, IN 47907
{mjohnson,harper}@ecn.purdue.edu

ABSTRACT

Large vocabulary speech recognition applications can benefit from an efficient data structure for representing large numbers of acoustic hypotheses compactly. Word graphs or lattices generated by acoustic recognition engines are generally not compact and must be post-processed to keep lattice sizes small; however, algorithms designed for this task need to reduce the size of the lattice without either eliminating hypotheses or distorting their relative acoustic probabilities. In this paper, we will discuss the relevant criteria for measuring graph size, compare the advantages of two different structures for graphs, and introduce a new data structure and compression algorithm which give additional graph compression and maintain exact hypothesis path scores by storing probability information on both nodes and arcs within the graph.

1. INTRODUCTION

Many recognition systems use word lattices or graphs as mechanisms for representing sentence hypotheses and interfacing with additional knowledge sources [1, 3, 8, 9, 10, 11]; however, the exact form of such lattices varies. Generally, lattices are represented as directed acyclic graphs. In some systems [8, 10], arcs represent words and nodes represent specific points in time, while in other systems [3, 11], nodes represent words and arcs represent transitions between words.

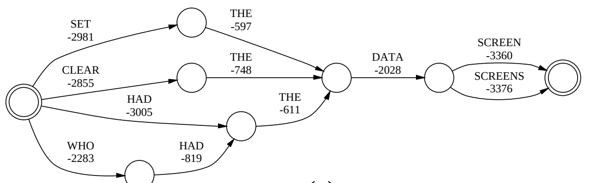
The terms word lattice and word graph are commonly interchanged as well, to refer to either of the above graph structures. For convenience, we will arbitrarily differentiate between the definitions as follows:

Lattice or word lattice: Words are represented by arcs and arc weights correspond to acoustic likelihood scores (usually log probabilities).

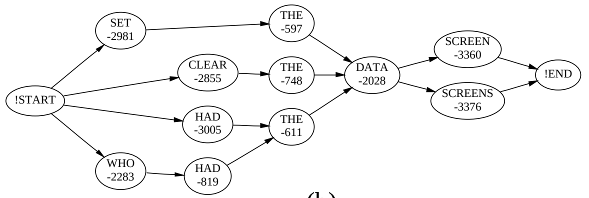
Word graph: Words are represented by nodes and node weights correspond to acoustic likelihood scores.

This research is supported by the National Science Foundation under Grant No. IRI-9704358.

Figure 1 illustrates the difference between a lattice and a word graph. Note that lattices and word graphs have a one-to-one correspondence, so that each is derivable from the other. Converting a lattice into a word graph is direct; each lattice arc becomes a node and each lattice node becomes multiple arcs. Conversion from a word graph into a lattice is only slightly more difficult and is accomplished using a similar process.



(a)



(b)

Figure 1: (a) Lattice and (b) word graph structures

Measures for evaluating the quality of lattices or graphs vary. Recently, some attention has been given to using Finite State Automata (FSA) methods to determinize and then minimize the lattices. Extensions to standard FSA techniques such as sub-set construction and state minimization have been developed [2, 7] which retain arc weights and therefore hypothesis rankings. This approach results in a lattice which has the minimum number of states possible given the constraint that the lattice be deterministic.

For applications which need to search the lattice for specific sentence hypotheses, a lattice optimized in this fashion gives fast performance times, since the requirement of determinism ensures that no branching is required to perform a string match (in the forward direction) and that time complexity is therefore linear with respect to the number of words in a hypothesis. Similarly, searching for the single best hypothesis based on path weights is slightly faster for a deterministic lattice as well, since each path is guaranteed to exist only once and therefore the number of distinct paths is minimized. This effect tends to be rather moderate, though, since the number of unique hypotheses remains unchanged.

If the goal, however, is to perform some additional type of model processing on the lattice (such as weight rescoring based on alternative word or language models), the issue of whether the graph is deterministic is not generally significant, and criteria such as the number of word arcs may be far more important than the number of states.

For this research, word graph structures are being used as an interface mechanism between acoustic recognition and a Constraint Dependency Grammar (CDG) parsing system [3, 4, 5]. CDG parsing uses unary (single word) and binary (word pair) constraints to model syntactic and semantic relationships between elements of a sentence hypothesis, and the system's time-to-parse performance is therefore significantly affected by the number of word nodes in the graph. Our best measure of word graph quality is simply to minimize the overall number of word nodes, while maintaining the scored hypothesis rankings without distortion (i.e., maintaining the acoustic probability scores of all hypotheses without addition or deletion or paths). Any knowledge source based on application of word-level models to elements in a word graph or lattice will have an implementation time that is a directly proportional to the number of words present. The CDG system we use for parsing is polynomial with respect to the number of nodes.

Globally minimizing the number of words in a lattice or word graph is a difficult problem; in the following sections we will examine the problem in more detail and look at a simple algorithm which approximates the global solution.

2. NODE COMPRESSION ALGORITHM

More formally, we will define a word graph as the set (V, E) of nodes and arcs which make up a Directed Acyclic Graph (DAG). Each node $v_i \in V$ has associated with it an alphanumeric value WORD(v_i) and a negative floating point weight SCORE(v_i). Without loss of generality, we can assume a single starting and ending node (since such a graph can always be constructed), which we will designate by the word strings "!START" and "!END". The set of arcs incoming to a node v_i is referred to as the predecessor or "prev" list, PREV(v_i), and the set of outgoing arcs is its successor or "next" list, NEXT(v_i).

A *path* through the word graph is defined as a word set (WORD(v_1),WORD(v_2),...WORD(v_n)), where WORD(v_1) = !START and WORD(v_n) = !END. The *score* of a path p is the sum SCORE(v_1) + ... + SCORE(v_n), the sum of the weights along the path.

We will define the compression of a set of nodes within a word graph as any transformation where the given set is replaced by a smaller set, such that all paths and path scores through the set are unchanged. Note that the case where all node scores are zero is equivalent to removing the requirement for identical path scores, so that only path equivalence is considered.

The graph minimization criteria can thus be formulated in terms of maximal compression. Since it is clear that no nodes containing different words can ever be compressed without altering graph paths, the compression problem can be divided into k subproblems, where k is the number of unique words within a word graph. Minimality is achieved when each subset of word-equivalent nodes is compressed to a minimum number of nodes. Since correct compressions cannot alter word paths or scores, all subsets are independent with respect to word paths (although not with respect to prev and next lists, since node-level connections between subsets are altered by compressions).

A compression from one node set to another can be approximated by a sequence of two-node compressions. Such a sequence of two-node compressions is not fully equivalent to a general-case set compression, since examples can easily be constructed where no two-node compressions are possible but overall set compression is possible. For this to happen, however, requires a complex arc arrangement where the prev and next lists of some nodes are subsets of the combination of prev and next lists from other word-equivalent nodes. The characteristics of our word graph structure, converted from an acoustically-generated lattice, tends to limit these cases. The original lattice-converted graphs, in fact, must always have either identical or disjoint prev and next lists, strengthening the validity of our approximation.

Looking specifically at the two-node case without node weights, a compression results in a new node such that the prev and next lists are unions of the prev and next lists of the two original nodes. This union results in a multiplication effect that increases the total number of paths represented. Examining this effect, it can be shown that a compression between a pair of nodes is valid only if the two nodes contain the same word and meet one of the following criteria:

- The prev lists are identical.
- The next lists are identical.
- The prev and next lists of one are subsets of the prev and next lists of the other. (As noted above, this cannot occur in word graphs which are generated directly from word lattices.)

When we also consider the confounding factor of nodes weights, the situation becomes much more difficult. If the scores associated with a pair of words are different, a direct compression would result in alteration of some hypothesis scores and could therefore cause a change of hypothesis rankings.

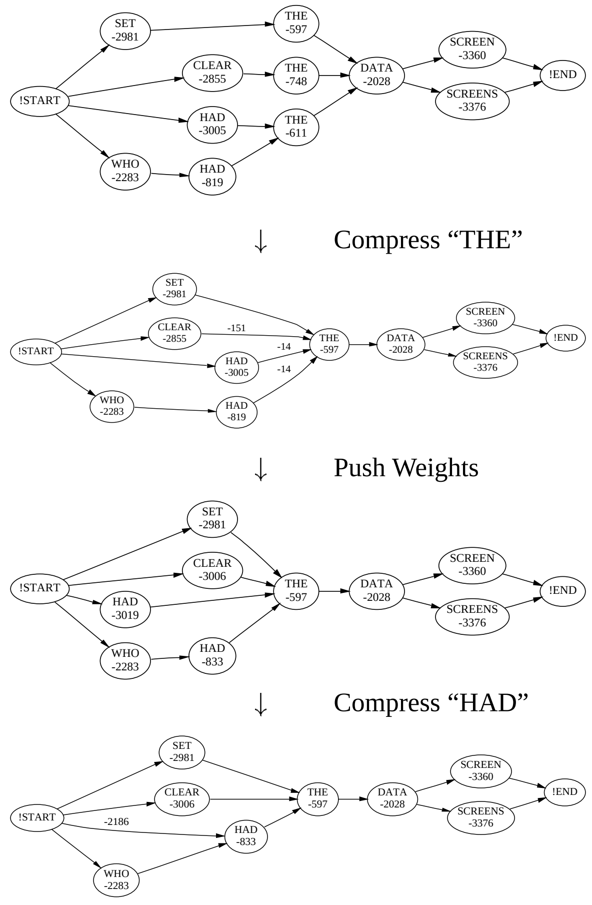


Figure 2: Compression example

To preserve paths and their scores during the compression process, we use a modified word graph structure that stores weights on arcs between words as well as within the word nodes themselves. By storing the difference between word scores as arc weights, we can compress word nodes which have two different scores into a single node. The criteria for node compression must also be modified to handle arc weights: to be compressible, two nodes must have identical prev or next lists with identical arc weights.

An example of compression using this new structure in shown in Figure 2, and abbrieviated pseudocode is shown in Figure 3.

Weights on nodes and arcs can be redistributed arbitrarily, by adding any amount to a node weight and subtracting the same amount from all of the prev arcs or all of the next arcs. Using this technique as a post-processing step after each compression, we can eliminate arc weights whenever possible to maximize the possibility of additional future compressions. We call this process “pushing” the weights, since it tends to push differences between weights further along the graph.

```
while (number of nodes has changed) {
  if (  ((PREV(i) == PREV(j)) AND
          (PREV_WEIGHTS(i) == PREV_WEIGHTS(j)))
     OR ((NEXT(i) == NEXT(j)) AND
          (NEXT_WEIGHTS(i) == NEXT_WEIGHTS(j)))) {

     Add arc weights of -|SCORE(i)-SCORE(j)| on
         the node with min(SCORE(i),SCORE(j)).
     Set SCORE(i) = max(SCORE(i),SCORE(j)).
     Set PREV(i) = Union(PREV(i),PREV(j)) and
         NEXT(i) = Union(NEXT(i),NEXT(j))
     Replace references to j with i.
     Push arc weights into neighboring node
         if all arcs are equally weighted.
  }
}
```

Figure 3: Compression algorithm

The compression algorithm is applied among pairs of nodes within the graph, continuing until no further compressions are possible. For simplicity of implementation, we do prev compressions first, followed by next compressions. It should be noted that there are rare cases where the order of compression can make a difference among a set of nodes with respect to the final remaining number of nodes. This is demonstrated in Figure 4, where it can be seen that doing prev followed by next compression results in two nodes, while next followed by prev compression results in three nodes. Given a set of identical word nodes with overlapping prev and next lists, it is possible to decide which approach is better for that particular case, but since the choice could in fact vary for different node sets within the same graph, and since the occurrence is rare, we did not implement that algorithm for this experiment.

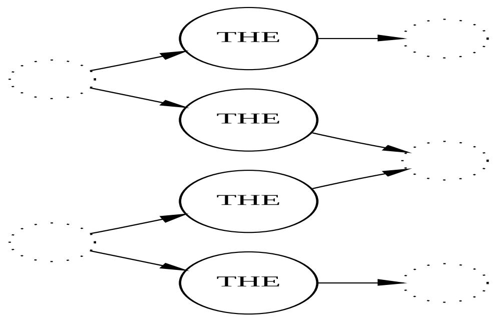


Figure 4: Case where compression ordering alters result

Table 1: Compression results

Graph Type	Avg. word nodes
Original Graph	75.15
Compr. Graph	28.62
Compr./Parsed Graph	12.07
Det. Graph	60.77
Det./Min. Graph	56.07
Det./Min./Compr. Graph	29.10

3. RESULTS

Using the Resource Management corpus, a naval corpus with a vocabulary of approximately 1000 words, we compared the number of word nodes resulting from using FSA determinization and minimization [2] versus using our compression algorithm. Our front-end acoustic engine was an HMM tri-phone system with an integrated word-pair language model, built using Entropic's HTK package [6]. The test set consisted of 1200 sentences, with a word-error rate of 5 percent, a 1-best sentence accuracy of 72.1 percent, and a word graph coverage accuracy of 95.1 percent. The average uncompressed word graph size was 75.15 nodes.

Compression Results are shown in Table 1. Our node compression algorithm resulted in a average compressed word graph size of 28.62 word nodes, compared to an average of 56.07 word arcs in a determinized minimized lattice (recall that there is a one-to-one correspondence between the number of word graph nodes and the number of lattice arcs). This represents a 48.9 percent reduction in the number of word graph nodes compared to FSA-based methods.

After CDG parsing was applied, only 12.07 nodes per graph and 7.02 paths per graph remained on average. Sentence accuracy was increased from 72.1 to 86.2 percent. Parsing times for the determinized minimized graphs were about double those of the corresponding compressed word graphs.

Since a fully optimal solution is unknown, it is difficult to quantify how close to minimal the compression results are. To estimate this, we used verification techniques which examined the compressed graphs and applied simple criteria to identify node sets which were guaranteed to be minimal, such as all single and dual sets of word-identical nodes (since single nodes are clearly minimal and our algorithm is minimal for two-node sets as well). Using this approach, we were able to immediately verify that 61.7 percent of the word graphs generated were fully minimal. A sampling of the remaining word graphs, analyzed by hand, was unable to identify any non-minimal cases, even for some of the larger graphs.

4. CONCLUSIONS

We have shown that the straightforward measure of word graph size can sometimes be a better minimization criteria than determinization and state minimization, and demonstrated a simple compression algorithm which is near minimal with respect to this criteria. Our algorithm results in word graphs averaging about half the size of those generated by determinization/minimization algorithms. The time savings achieved by having smaller word graphs can be extremely significant for supporting additional post-acoustic knowledge sources.

5. REFERENCES

[1] Xavier Aubert and Hermann Ney. Large vocabulary continuous speech recognition using word graphs. *Proc. ICASSP*, pages 49–52, 1995.

[2] AT&T Corporation. FSM library version 3.6, 1999. http://www.research.att.com/sw/tools/fsm.

[3] Mary P. Harper, Leah H. Jamieson, Carla H. Zoltowski, and Randall A. Helzerman. Semantics and constraint parsing of word graphs. *Proc. ICASSP*, pages 63–66, 1993.

[4] Mary P. Harper, Michael T. Johnson, Leah H. Jamieson, Stephen A. Hockema, and Christopher M. White. Interfacing a CDG parser with and HMM word recognizer using word graphs. *Proc. ICASSP*, 1999.

[5] Michael T. Johnson, Mary P. Harper, and Leah H. Jamieson. Interfacing acoustic models with antural language processing systems. *Proc. ICSLP*, 1998.

[6] Entropic Cambridge Research Laboratory Ltd. HTK version 2.1, 1997.

[7] Mehryar Mohri and Michael Riley. Finite-state transducers in language and speech processing. *Computational Linguistics*, 1997.

[8] Martin Oerder and Hermann Ney. Word graphs: An efficient interface between continuous-speech recognition and language understanding. *Proc. ICASSP*, pages II–119–122, 1993.

[9] S. Ortmanns and H. Ney. A word graph algorithm for large vocabulary continuous speech recognition. *Computer Speech and Language*, pages 43–72, 1997.

[10] F. Richardson, M. Ostendorf, and J.R. Rohlicek. Lattice-based search strategies for large vocabulary speech recognition. *Proc. ICASSP*, 1995.

[11] Fuliang Weng, Andreas Stolcke, and Ananth Sankar. Efficient lattice representation and generation. *Proc. ICSLP*, 1998.