The Effect of Pruning and Compression on Graphical Representations of the Output of a Speech Recognizer

Yang Liu, Mary P. Harper[*], Michael T. Johnson[†], Leah H. Jamieson[*]

February 14, 2002

Abstract

Large vocabulary continuous speech recognition can benefit from an efficient data structure for representing a large number of acoustic hypotheses compactly. Word graphs or lattices have been chosen as such an efficient interface between acoustic recognition engines and subsequent language processing modules. This paper first investigates the effect of pruning during acoustic decoding on the quality of word lattices and shows that by combining different pruning options (at the model level and word level), we can obtain word lattices with comparable accuracy to the original lattices and a manageable size. In order to use the word lattices as the input for a language processing module, they should preserve the target hypotheses and their scores while being as small as possible. In this paper, we introduce a word graph compression algorithm that significantly reduces the number of words in the graphical representation without eliminating hypotheses or distorting their acoustic scores. We compare this word graph compression algorithm with several other lattice size-reducing approaches and demonstrate the relative strength of the new word graph compression algorithm for decreasing the number of words in the representation. Experiments are conducted across corpora (RM and WSJ) and vocabulary sizes (WSJ 5k and 20k) to determine the consistency of the pruning and compression results.

1 Introduction

Word lattices are often chosen as the interface between an acoustic recognizer and a subsequent processor using a more complex language model (LM) or more specific acoustic model because of their ability to compactly represent the large number of utterance hypotheses produced by the recognizer. To be an efficient representation, a lattice should be as small as possible while maintaining its accuracy. This size/accuracy quality can be obtained by

[*]{yangl, harper, lhj}@ecn.purdue.edu
[†]mike.johnson@marquette.edu

using two potentially complementary approaches. First, pruning options during acoustic decoding can have a significant impact on the quality of word lattices. Second, the lattices produced by the acoustic recognizer can be further post-processed to reduce their size and possibly increase their accuracy. In this paper our goal is a thorough investigation of the effect of acoustic pruning and post-processing pruning/compression methods on the quality of lattices.

In large vocabulary continuous speech recognition systems, pruning is indispensable for reducing computational effort and improving efficiency. Speech recognition is a search problem with its search space being dependent on the network imposed by the language model. Pruning can often reduce the search space without eliminating the target sentence or word hypotheses. In almost all of the current speech recognition systems, effective pruning strategies have been employed. Odell [18] has systematically studied the effect of pruning on the computational efficiency and accuracy of the recognizer. However, that research focused only on the 1-best recognition result. In this paper, careful study is made of the impact that pruning during acoustic decoding has on the properties of word lattices. Preliminary results [9] on the Resource Management task suggest that we can obtain a good balance between lattice size and accuracy by carefully choosing pruning options. We will further verify this across corpora and vocabulary sizes. Lattice density, lattice sentence accuracy, and lattice word error rate are used to measure the quality of the word lattices along with the standard 1-best word error rate. We also compare the word lattice to the N-best list representation over a variety of pruning parameters.

In a multi-pass speech recognition system, an efficient representation of the large number of utterance hypotheses is beneficial to the system. A word lattice is often used as the data structure for storing the sentence hypotheses [4, 16, 20, 24]; however, it has been defined in a variety of ways by the speech community. A word lattice represents the raw output of the acoustic recognizer, that is, it is a complete record of all word tokens which are not pruned during the recognition process. It may include many similar or even identical paths with slight differences in timing alignments. In some systems [19, 20, 22], arcs represent words and nodes represent specific points in time, while in other systems [27], nodes represent words and arcs represent transitions between words. Here we assume that the edges in a lattice are used to represent words and their likelihood scores (usually log probabilities). Figure 1 illustrates a word lattice under this definition.

The word lattices initially produced by the acoustic recognizer even with pruning are often large and highly redundant. To save computational effort in a subsequent language processing module, the size of the representation passed to it should be as small as possible. One way to further reduce the size of the lattices produced by the acoustic recognizer is to compress or prune them after decoding. Some algorithms have been developed for this purpose [10, 13, 14, 25]. Since most language processing algorithms applied to word lattices run in polynomial time with respect to the number of words in the representation, we have designed a new word graph compression algorithm to reduce the number of words in

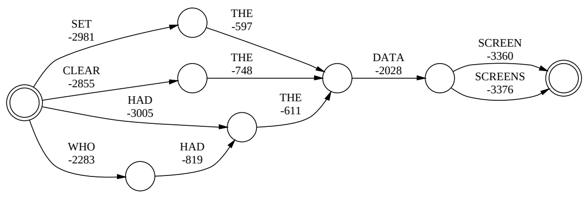


Figure 1: An example of a word lattice.

the graphical representation while maintaining the scored hypothesis information. In this paper, we will compare this algorithm with several other methods that reduce the size of lattices after decoding and will also investigate the combinations of some of these methods.

The structure of this paper is as follows. In the next section, we briefly introduce the recognition tasks and the recognition system architecture. Section 3 describes how pruning during acoustic decoding affects the quality of word lattices across recognition tasks. In section 4, we first introduce the new word graph compression algorithm and several other lattice post-processing methods, and then compare and evaluate them individually and in combination. A summary of our findings appears in section 5.

2 Speech Recognizer

2.1 Recognition Tasks

To investigate the two factors that affect the quality of a graphical representation for the output of a speech recognizer, we chose two speaker independent continuous recognition tasks: the Resource Management and Wall Street Journal tasks. These corpora are obtained from LDC [1].

The Resource Management (RM) [21] contains utterances concerning the management of Naval resources. There are 2,845 distinct sentences derived from 900 different sentence templates based on interviews with naval personnel familiar with naval resource management tasks. Due to domain specificity of the task, the vocabulary is approximately 1,000 words and the task perplexity is fairly low. A word pair grammar is provided with the data and is used in this paper for constrained recognition. We also use the extended Resource Management corpus (RM2) consisting of a different set of sentences than RM but using the same lexicon and spoken by a different group of speakers.

The Wall Street Journal (WSJ) corpus is larger and more varied than RM. The prompting text are excerpts from the Wall Street Journal. In addition to the waveform data, WSJ contains complete orthographic transcriptions of the speech data and bigram and trigram language models for the text data from which the prompting text was taken. Official evaluations were conducted in Nov'92, Nov'93 and Nov'94, each of which has a variety of test conditions. In our experiments, we use 92 and 93 5K and 20K-vocabulary recognition tasks.

2.2 Speech Recognition Architecture

The acoustic portion of the system is a multiple-mixture triphone HMM constructed using HTK [2]. Initial signal processing was implemented using a 24-bank Mel Scale filter to compute 12 cepstral coefficients plus a normalized energy coefficient. Hamming windows were used on 25 millisecond acoustic frames at a 10 millisecond frame rate. Delta and delta-delta cepstrals were also computed, giving a total feature vector size of 39 elements.

Initial training was done using flat-start monophone models, followed by embedded Baum-Welch re-estimation. Monophone models were split into context-dependent triphones, and then clustered into tied groups using a decision tree state-tying approach [29], followed by re-estimation. Next, observation distributions were split from single Gaussians into Gaussian Mixture Models with eight mixtures[1]. A token-passing implementation [28] of the Viterbi algorithm was used during recognition, the output of which is a large word lattice for each test sentence.

3 Pruning During Decoding

A high quality word lattice is very important as an input to the subsequent language processor to the system performance. Pruning mechanisms are used during the recognition process to make it computationally feasible. Pruning not only has a significant impact on system speed, but also affects the quality (accuracy and size) of word lattices produced by the acoustic recognizer. If no pruning is done during the acoustic recognition process, the lattice can be highly accurate but also exorbitantly large. In this section, we focus on the impact of different pruning options on the quality of word lattices, and investigate the tradeoffs among a variety of measurements.

3.1 Pruning Methods

Pruning can be applied at several different levels during acoustic recognition. Generally the higher the level, the greater the available knowledge, the tighter the pruning can be without eliminating the correct hypothesis. In the experiments described in this section, we use three pruning options that are described below.

- Beam width pruning is applied at the model level. In this method, the total number of active models is limited by a beam width mechanism based on the difference between the log probability of each active model and the current maximum log probability.

[1]On the WSJ task, we used 16 Gaussian mixtures for silence and short pause.

- Maximum active models pruning is also applied at the model level. This method enforces a fixed limit on the number of models allowed to be active during recognition. Since the possible number of active models could be much greater than the average number of active models, setting a maximum limit can reduce the memory consumed.
- Word-end pruning is also a beam width mechanism that considers only word-end nodes within the recognition network, thus allowing pruning to happen at the word level rather than the phoneme model level.

We will investigate the effect of these pruning methods and how they interact.

Another factor affecting the search space is the grammar scale. Pruning is typically based on the combined score of both the acoustic likelihood and the language model score weighted by this grammar scale; hence, we also investigate the effect of the grammar scale.

3.2 Performance Measures

We use the following measurements to evaluate the effect of pruning.

- Computational effort (or the system speed) is measured by the average number of active models per frame during recognition.
- The lattice complexity is measured by the lattice word density, which is defined to be the number of words in the lattice divided by the actual number of words uttered.
- The lattice sentence accuracy is the percentage of lattices for which the test sentence appears as a valid path.
- The lattice word error rate is computed by determining the sentence in the lattice that best matches the uttered sentence [20]. This measure provides a lower bound of the word error rate for this lattice.
- The standard word error rate of the 1-best recognition result is also shown as a measurement of the recognition performance.
- The information gain obtained by the word lattices over the N-best lists is defined to be the number of test sentences for which the word lattice contains the correct sentence but the N-best list does not (N is fixed to be 10 in the pruning experiments). Since the absolute lattice accuracy is already measured by the lattice sentence accuracy and the lattice word error rate, this relative information gain is somewhat redundant; however, it is included here to determine whether a word lattice contains more information and is more compact than an N-best list.

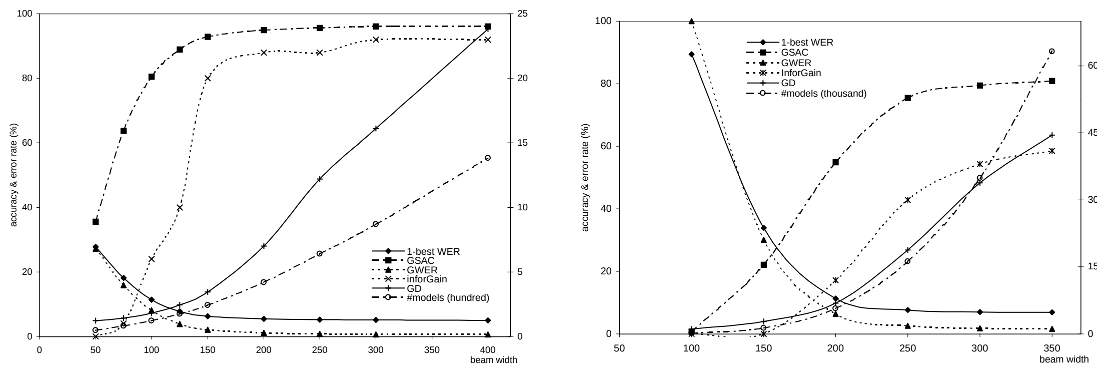


Figure 2: The effect of beam width pruning on RM (left) and WSJ (right) task.

3.3 Experimental Setup

To investigate the effect of acoustic pruning on the quality of word lattices and system computational effort, we chose RM and WSJ 5K-vocabulary recognition tasks. These two tasks were chosen because they are large enough to provide the characteristics of large vocabulary tasks and small enough to make systematic investigation possible.

On the RM task, the speaker independent training set includes 109 speakers and a total of 3,990 high-quality recorded sentences. The test set is composed of February 1989, October 1989, February 1991, and September 1992 evaluation sets, each of which has 10 speakers and 30 sentences per speaker, for a total of 1,200 test sentences. The provided word pair LM is used to construct recognition network for this task. On the WSJ 5K-vocabulary task, we use section WSJ0 SI84 (short term speaker-independent) for training the acoustic model. This consists of 7,193 sentences from 84 speakers for a total of approximately 12 hours of speech. The test set consists of the Nov'92 set, which includes 330 sentences containing 5,353 words spoken by 8 speakers. The provided bigram LM is used to constrain recognition for WSJ.

On each task, we modify one pruning parameter and fix the others to determine the effects of each pruning option. The other pruning variables are fixed to values that are chosen so that they would seldom cause search errors on their own. Additionally, the experiments performed on the two tasks were made as similar as possible in order to investigate the impact of the recognition task and its vocabulary size.

3.4 Pruning Results

3.4.1 Beam Width Pruning

Figure 2 shows the effect of beam width pruning on the RM and WSJ tasks. For this experiment, maximum model pruning is turned off, word-end pruning beam is set to 200,

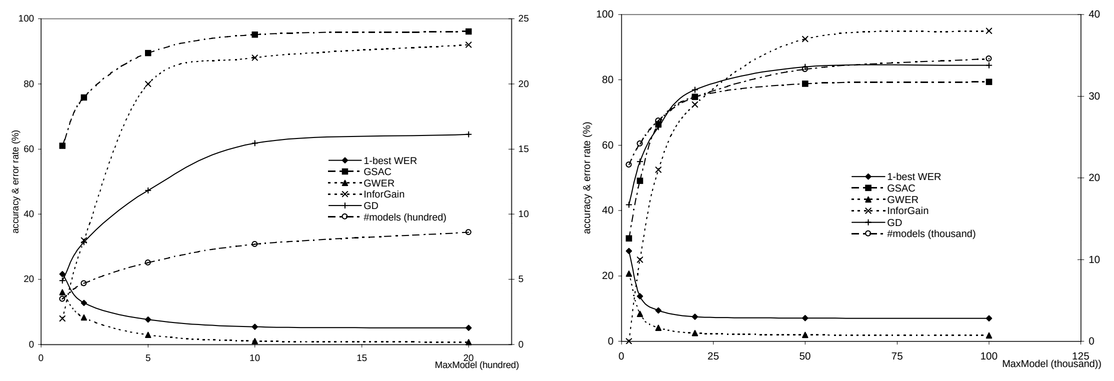


Figure 3: The effect of maximum model pruning on RM (left) and WSJ (right) task.

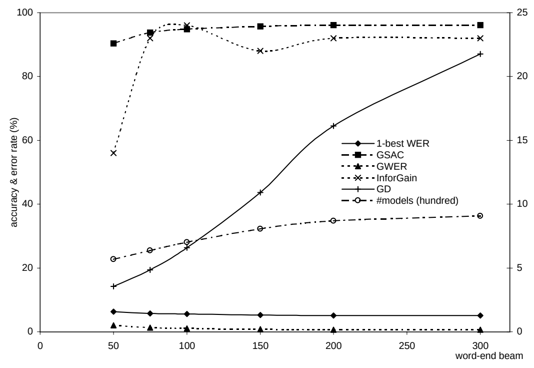
and the embedded LM scale is set to 7 for RM and 16 for WSJ[2]. In the two graphs of Figure 2 (as well as the following figures), the left Y-axis represents lattice word error rate (GWER), lattice sentence accuracy (GSAC), and 1-best word error rate (WER); the right Y-axis is used for the other three measures: lattice word density (GD), the average number of active models (#models, with one hundred used as the unit for RM and one thousand for WSJ), and information gain (InforGain).

The two graphs in Figure 2 show similar patterns. When the value of the pruning variable is increased, the number of active models, lattice accuracy, lattice size, and information gain all increase in both tasks. However, since WSJ is a more difficult task than RM, the beam width needs to be larger for that task to avoid search errors.

Increasing beam width significantly increases computational effort and lattice size over the entire range of settings, but only reduces accuracy when it is below some specific value that depends on the particular task. As can be observed from the graphs in Figure 2, as the beam width increases from its lowest value (50 for RM and 100 for WSJ), the increase in accuracy is quite dramatic, but the increase in lattice word density and average number of active models is much slower. However, as the beam width increases further to a value, for example 150 for RM and 250 for WSJ, then the change in accuracy is very slight, but the lattice word density and the average number of active models increase almost linearly.

Consider the graph for the WSJ task; we observe that using a beam of 250 can double the decoding speed compared to the beam of 300, and the search errors caused by decreasing the beam are very slight. However, further reducing the beam width decreases the recognition performance (e.g., on WSJ a beam of 200 increases the error rate significantly; lattice word error rate drops to 6.4% compared to 1.79% using a beam of 300). Our results suggest that we can obtain a high quality lattice (good accuracy and moderate size) using limited

[2]The grammar scale for RM is selected based on the results from [7], and the grammar scale for WSJ is selected to minimize the 1-best WER of the 92 development set for the 5K-vocabulary task.



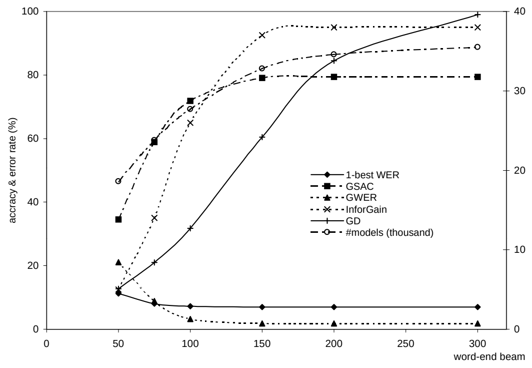


Figure 4: The effect of word-end pruning on RM (left) and WSJ (right) task.

computational effort by intelligently selecting beam width.

3.4.2 Maximum Active Model Pruning

Results from varying maximum active models are shown in Figure 3. In this experiment, beam width is set to 300, word-end pruning beam is set to 200, the LM scale is set to 7 for RM and 16 for WSJ. The effect of maximum model pruning is similar for these two tasks; increasing the value of maximum active models improves accuracy and increases lattice size and the number of active models in both cases. Unlike beam width pruning, increasing maximum model pruning beyond some task-dependent value has a limited effect on lattice size, accuracy, and system speed. From the graphs, we can observe that almost all the measurement values are flat after that value is reached.

Since the number of the frames that have active models several times more than the average number is quite small, when the maximum model limit is set to be several times the average number, it affects the performance only slightly. For example, when the maximum model limit is set to 2K for RM and 100K for WSJ, the recognition results are the same as when maximum models pruning parameter is turned off.

The maximum model limit does not affect the computational effort as significantly as beam width pruning does; however, reduction of the maximum limit to some small value can decrease the accuracy. Hence, lowering the maximum model value to speed up the system is not a good design choice.

3.4.3 Word-end Pruning

Figure 4 shows the effect of word-end pruning. In this case, the model beam width is set to 300, maximum model pruning is turned off, and grammar scale is set as in the previous experiments.

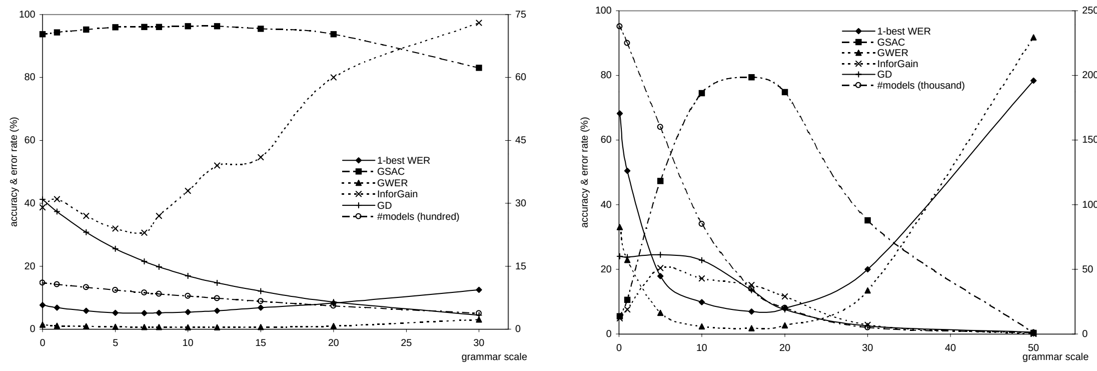


Figure 5: The effect of grammar scale on RM (left) and WSJ (right) task.

Again we observe similar patterns in these two graphs. Word-end pruning has a less dramatic effect on the lattice accuracy and computation compared to beam width pruning and maximum model pruning for both tasks. However, word-end pruning has more significant effect on the WSJ than on the RM task because WSJ has a larger vocabulary and requires more word hypotheses to obtain a good accuracy level.

When the word-end pruning setting is increased beyond a certain point for each task, the lattice size increases significantly (approximately linearly), without any improvement in accuracy. This is because word-end pruning is an important factor in deciding the number of word hypotheses in the lattices produced. Increasing this beam width can add many hypotheses that are very unlikely, which is ineffective for increasing accuracy. For example, on the WSJ task, the accuracy is stable from the value of 150 and up; as the setting increases beyond 150, the size of the lattice increases without any additional gain in accuracy; when the word-end beam is reduced to a value less than 75, a significant decrease in recognition performance is observed.

3.4.4 Grammar Scale Factor

Since the pruning is performed based on the combination of the acoustic score and language model score weighted by the grammar scale, the choice of the grammar scale also has an important impact on the lattice quality. Figure 5 depicts the effect of the grammar scale. For this experiment, the beam width is set to 300, maximum model pruning is turned off, and word-end beam width is set to 200 for both tasks.

Increasing the grammar scale (and thus the LM score) can increase system speed and reduce the lattice size because fewer hypotheses would be preserved. However, only when the grammar scale is set to a suitable value, can it yield accurate recognition results. When the grammar scale is too small, the impact of the LM is slight, so it cannot constrain the search space adequately. When it is too large, the acoustic impact is weakened and the

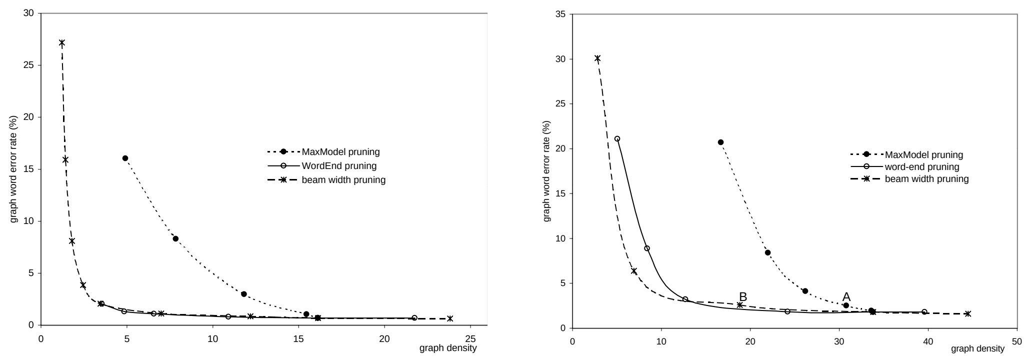


Figure 6: The relationship between graph word error rate and graph density on RM (left) and WSJ (right) task.

error rate increases.

There are some differences in the patterns observed from the graph for each task. The accuracy and error rate curves change more dramatically on the WSJ than the RM task, and the "peak" of the curve on WSJ is more pronounced. One reason for this discrepancy is that on the RM task, we use a word pair grammar to constrain the search space; whereas, on the WSJ task we use a well-trained bigram language model. On the other hand, because RM is an easier task, there is less acoustic ambiguity, so even when the grammar scale is 0, the degradation in recognition performance is lower than the WSJ task, which requires more help from the language model to eliminate acoustic ambiguity. So the grammar scale has a more significant impact on the search space, accuracy, and lattice size for the WSJ task.

3.4.5 Combined Pruning

One goal in this research is to maximize the lattice performance (high lattice accuracy and small lattice size) using limited system computation time. It is clear from the previous experiments that each pruning option has a different impact on the system performance, and that these pruning variables interact. Understanding the individual impact of each pruning variable and their interactions is important for obtaining lattices with the desired characteristics. For example, choosing to reduce the maximum model limit or a word-end pruning value to speed up the system can sometimes be more harmful to the lattice accuracy than lowering the beam width value.

We need to set each pruning value large enough to obtain a highly accurate lattice for the input to the language processing module. However, when each pruning value increases beyond the value that achieves a "peak" accuracy, they impact system speed and lattice size differently. Increasing beam width lowers system speed and increases lattice size. Increasing

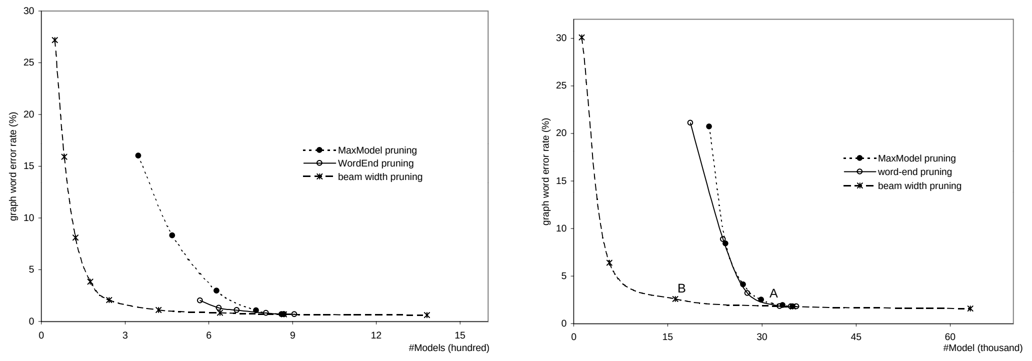


Figure 7: The relationship between graph word error rate and the number of active models on RM (left) and WSJ (right) task.

the maximum model pruning value only affects the speed indirectly by consuming more memory with little effect on the lattice size. Increasing the word-end pruning value leads to larger lattices.

Since our focus is to generate high quality lattices with limited computation, to further investigate the relationship among computational effort, lattice accuracy, and lattice size, we will depict the previous results in another way. We want to determine how lattice word error rate changes along with lattice size and the average number of active models, and how these properties are affected by pruning options. Next, we re-plot the results from the previous experiments to investigate the interaction, first, of the lattice word error rate and lattice word density, and then lattice word error rate and the number of active models[3].

Figure 6 depicts the relationship between the lattice word error rate and its word density for RM and WSJ tasks. It is important to note that larger lattices do not always give better lattice accuracy. With poorly chosen values for the pruning variables, it is possible to end up with larger lattice size and lower lattice accuracy, and this is true for both tasks. Take point A and B shown for the WSJ task in Figure 6 as an example. A has lattice word density of 30.8 and lattice word error rate 2.5%; B has density of 18.8 and lattice word error rate 2.6%.

Figure 7 shows the relationship between lattice word error rate and computational effort measured by the average number of active models. We can see that more computational effort does not always produce better accuracy, and similar recognition results can be obtained with varying computational effort. For example, on the WSJ task, A and B have a similar lattice word error rate (2.5% and 2.6%), but the number of active models of A (29.9k) is nearly twice that of B (16.2k), suggesting that the computational cost of A is

[3]For the maximum model pruning, beam width is 300, word-end beam is 200. For word-end pruning, maximum model limit is turned off, model beam width is 300. For beam width pruning, maximum model limit is turned off, word-end beam is 200. LM scale is 7 for RM and 16 for WSJ in all the cases.

much higher than that of B.

3.5 Pruning Conclusions

This cross-corpus investigation allows us to determine the effects of each pruning variable and their interaction on system performance. This study should help us to more generally choose pruning options for obtaining high quality word lattices for a post-processing module. Moderate experimentation with a development set should be sufficient to obtain high quality lattices with reasonable computational effort for a new task.

The pruning performed at the model level (beam width and maximum model pruning) must be loose enough to preserve the possible hypotheses to maximize lattice accuracy. For a large vocabulary recognition task, a beam width of 300 is a good starting point. Experimenting with values slightly above and below this value will reveal the appropriateness of this value. Beam width is the most important pruning option affecting the tradeoff among the accuracy, the size of the lattices, and the system computational effort. Maximum model pruning could be turned off initially and then based on the number of average active models for each frame, it should be set to several times the average number to reduce memory consumption and increase system speed without decreasing lattice accuracy (see Figure 3).

Word-end beam pruning has the least impact on the lattice accuracy, since the pruning parameter could be set to a tighter value and still retain the correct hypotheses; however, the impact of this pruning variable on the lattice size is comparable to pruning at the model level. A word-end beam width of 200 or 300 is likely to be good enough for most large vocabulary recognition tasks.

The grammar scale is also important to intelligently reduce the search space, and this is particularly true when a better language model is used to constrain the search space, as we have found on the WSJ task. The grammar scale must be optimized based on the task and the quality of the LM using a development set to ensure good performance [26].

The experiments thus far have also indicated that the word lattice contains more correct sentence hypotheses (measured by the information gain) and thus is a more compact representation than the N-best list [17, 23] as an interface between the recognizer and the subsequent language processing module.

4 Reducing Lattice Size After Decoding

Language processing modules are often applied to word lattices after decoding. These modules can run in time polynomial in the number of words in the input, so minimizing lattice size without sacrificing accuracy can have a big impact on the accuracy and running time of the language processing module. Pruning during decoding is only a first step toward minimizing lattice size. Post-processing the lattice can further reduce its size while maintaining accuracy.

Lattice post-processing methods can be divided into two categories based on whether they are lossless or not. Lossless algorithms, which are also called compression algorithms, maintain all sentence paths and their scores in the lattices, thus preserving related ranking information. Lattice compression is feasible because some information is redundant or unnecessary for a language processing module. For example, identical sub-paths may be included more than once in the lattice due to differences in word starting and ending times. By contrast, lossy algorithms remove some unlikely hypotheses in the lattices and also may add new path hypotheses to the lattices.

In section 4.1, we first introduce a new lossless word graph compression algorithm that we have designed to reduce the number of words in the representation without eliminating hypotheses or distorting relative acoustic probability ordering. Section 4.2 describes several other lattice post-processing methods, including two lossy algorithms: forward-backward likelihood pruning and confusion network algorithms[4], and the lossless finite state determinization and minimization approach. In section 4.3 these lattice post-processing methods are evaluated individually and in combination across corpora and vocabulary sizes.

4.1 The Word Graph Compression Algorithm (WGC)

If a language processing module performs additional processing on the lattice such as weight rescoring based on alternative language models, criteria such as the number of word edges may be far more important than the number of states. Any knowledge source applied to words in a lattice will have an processing time that is directly proportional to the number of words present. The running time of parsing algorithms that process in parallel all the paths of a lattice or some other graph-based representation is significantly affected by the number of words in the representation (polynomial time complexity with respect to that number) [5]. In such a case, it is most important to minimize the number of words. Additionally, if the goal is to identify the best sentence hypothesis then it is important to maintain the scored hypothesis rankings without distortion (i.e., maintain the acoustic probability scores of all hypotheses without addition or deletion of unique paths). Hence, we have developed an algorithm that minimizes the number of words in a graph representation without modifying path scores. Preliminary results of this algorithm appear in [8] and the effectiveness of the compressed word graph for parsing efficiency was evaluated in [6].

Our algorithm operates on a word graph, which is a transformation of the word lattice produced by the recognizer. In a word graph, the words and their probabilities are represented as word nodes, and the connectivity originally represented by the states of the lattice are represented as edges between word nodes. Figure 8 depicts a word lattice along with its corresponding word graph. Note that there is a direct correspondence between the lattice and word graph. An advantage of the word graph representation is that scores

[4]Mangu [13] calls the confusion network approach a compression algorithm; however, based on our definition of compression, Mangu's algorithm is a lossy, not a compression algorithm.

can be associated not only with a word node, but also with the edges between word nodes. This representation will enable our word graph compression algorithm to more effectively reduce the number of words in the representation than finite state determinization and minimization of the lattice.

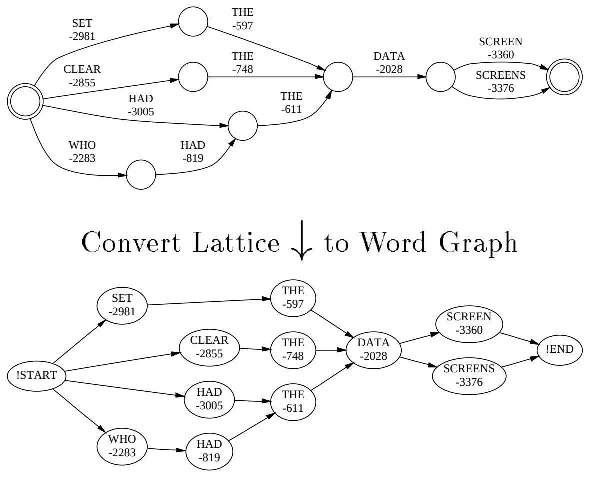


Figure 8: The correspondence between a word lattice and its corresponding word graph.

Word Graph Terminology:

More formally, a word graph is a Directed Acyclic Graph represented as a tuple $G = (V, E)$ comprised of a set of nodes and a set of directed edges. Each node with index $v_i \in V$ has an associated alphanumeric value WORD(v_i) and a floating point weight SCORE(v_i) that generally represents log probability data. Each edge (v_i, v_j) also has an associated floating point weight SCORE(v_i, v_j). When a word graph is initially created from a lattice, SCORE(v_i) for each vertex $v_i \in V$ is set to the probability on the corresponding word edge from the lattice, and SCORE(v_i, v_j) for each edge $(v_i, v_j) \in E$ is set to 0. In depictions of word graphs, if an edge has a score of zero, the value is not shown (as in Figure 8). Our algorithm keeps track of the incoming edges of node v_j by using PREV(v_j), which is a list of tuples containing the source node v_i of each edge (v_i, v_j) and the edge's score, SCORE(v_i, v_j). It also keeps track of the outgoing edges of node v_j by using NEXT(v_j), which is a list of tuples containing the destination node v_k for each edge (v_j, v_k) and SCORE(v_j, v_k). Note that PREV(v_l) = PREV(v_m) only if each tuple in PREV(v_l) has a corresponding tuple in PREV(v_m) that is equal to it, and vice versa.

Without loss of generality, we assume a single start and end node (such a graph can always be constructed), denoted *!START* and *!END*, respectively. Note that initially SCORE(!START) = SCORE(!END) = 0. A *path* in the word graph is defined as a tuple (v_0,v_1,v_2,...,v_n) such that v_0 is !START and v_n is !END, and (v_0,v_1), (v_1,v_2), ..., (v_{n-1},

v_n) are directed edges in E. The *score* of a path is the sum of the edge and node weights along the path:

$$s(v_0, v_1, \ldots, v_n) = \text{SCORE}(v_0) + \sum_{i=1}^{n}(\text{SCORE}(v_i) + \text{SCORE}(i-1, i))$$

A word path or *sentence* is defined as the set of words associated with a node path. Word paths may appear more than once in a word graph, but it is necessary only to preserve the maximum score associated with a sentence during word graph compression. Hence, the score of a specific sentence is the highest score of any path associated with that sentence.

Two Node Compression:

We define a compression on a word graph $G = (V, E)$ as any transformation where the set V is replaced with a smaller set V', such that the set of all sentences and their scores in G are unchanged. 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 given the connectivity of G without altering sentence scores. Since valid compressions do not alter sentences or their scores, the k subsets are independent. A compression from one node set to another is approximated here by a sequence of two-node compressions. In the two-node compression case, a pair of nodes, n_i and n_j, can be compressed only if WORD(n_i) = WORD(n_j) and one of the following is true:

1. PREV(n_i)=PREV(n_j)
2. NEXT(n_i)=NEXT(n_j)
3. $(\text{SCORE}(n_j) \geq \text{SCORE}(n_i) \wedge \text{PREV}(n_i) \subseteq_{\leq} \text{PREV}(n_j) \wedge \text{NEXT}(n_i) \subseteq_{\leq} \text{NEXT}(n_j))$
   $\vee\, (\text{SCORE}(n_i) \geq \text{SCORE}(n_j) \wedge \text{PREV}(n_j) \subseteq_{\leq} \text{PREV}(n_i) \wedge \text{NEXT}(n_j) \subseteq_{\leq} \text{NEXT}(n_i))$

For conditions one and two, set equality is at the level of the tuple; hence, the nodes must have exactly the same nodes either preceding or following them, each with the same edge weight. Without the equality of edge weights, it may be impossible to merge the two nodes while preserving the weights of the unique paths. We will describe how two nodes can be compressed based on condition one. Condition two is symmetric, and so we omit its description. Condition one holds if WORD(X_1) = WORD(X_2) and PREV(X_1)=PREV(X_2); hence, X_1 and X_2 can be merged into a single node. This can be done by creating a new node X with the same word string as X_1 and X_2 and setting its probability to the higher probability of the two nodes. Let SCORE(X_1) = x_1 and SCORE(X_2) = x_2, and assume that $x_1 > x_2$, then SCORE(X) = x_1. Next we must connect X into the word graph by carefully constructing PREV(X) from PREV(X_1) and PREV(X_2) and NEXT(X) from NEXT(X_1)

and NEXT(X_2) in a way that preserves the weight of the paths going through X instead of X_1 and X_2. Since PREV(X_1)=PREV(X_2), PREV(X) can be set to either set (or for convenience PREV(X_1) $\cup$ PREV(X_2)). Given that NEXT(X_1) may differ from NEXT(X_2), the construction of NEXT(X) requires care in order to preserve the path weights. Since $x_1 > x_2$, the paths leading out of X_1 require no adjustment; however, the paths leading out of X_2 must be adjusted to reflect the fact that the cost of the merged node is x_1 rather than x_2. Hence, we must subtract $x_1 - x_2$ from those paths; this can be achieved by subtracting that value from the edges leading out of X_2. Then, we can set NEXT(X) to be the union of NEXT(X_1) and NEXT(X_2), and delete duplicate edges with smaller edge weights. This has the effect of deleting duplicate paths with smaller weight. Now that X is connected into G, we can remove X_1 and X_2.

If there were no probabilities associated with the nodes and edges of a word graph, then it would also possible to compress two nodes X_1 and X_2 with WORD(X_1) = WORD(X_2) when PREV(X_1) $\subseteq$ PREV(X_2), and NEXT(X_1) $\subseteq$ NEXT(X_2) or when PREV(X_2) $\subseteq$ PREV(X_1) and NEXT(X_2) $\subseteq$ NEXT(X_1) without deleting paths or creating new ones. However, once there are node and edge scores associated with the word graph, we are unable to compress two nodes unless there is some way to preserve the probabilities of all the unique paths. The third condition allows for the compression of two nodes only if the node with the largest score has PREV and NEXT sets that each contain the PREV and NEXT (respectively) sets of the lower scoring node, and all of the edge weights coming into and exiting the lower scoring node have values equal to or less than the corresponding edge on the larger weight node. The operator $\subseteq_{\leq}$ ensures that each edge that appears in the first set also appears in the second and that its weight is less than or equal to the weight in the second set. For example, suppose that SCORE(X_1) $\geq$ SCORE(X_2) $\wedge$ PREV(X_2) $\subseteq_{\leq}$ PREV(X_1) $\wedge$ NEXT(X_2) $\subseteq_{\leq}$ NEXT(X_1). In this case, we can simply ignore the paths associated with X_2 since all of its paths are lower probability duplicates of paths associated with X_1. Without this additional condition, it becomes impossible to ensure that two nodes can be compressed into a single node while preserving path probabilities. For example, consider compressing the two D nodes in the top graph of Figure 9[5]. Here the D node with score 0.1 has PREV and NEXT sets that contain the PREV and NEXT sets of the D node with score 0.2. Even in this simple case where all the edge weights are 0, there is no way to assign probabilities to the nodes and edges after the node D is compressed to concurrently represent the probability of the paths containing node A or node G along with the paths containing both A and G. If we create a new D node with a score of 0.2, and combine the PREV and NEXT sets of the two original nodes, then we can represent the probabilities of the paths (A, D, E) and (A, D, F) by adding a weight of -0.1 to the edge (A, D), as well as

[5]To simplify this figure, it focuses on the two D nodes and their connectivity to preceding and following nodes; there may be additional nodes that precede and follow the other nodes that are not depicted in the figure.

the paths (B, D, G) and (C, D, G) by adding a weight of -0.1 to the edge (D, G); however, these two edge weights are incompatible with representing (A, D, G)'s weight of 0.3. When a word graph is created from an acoustically-generated lattice, all of the PREV and NEXT lists of a node are either identical or disjoint from each other because of the fact that the word edges entering a state in a lattice are only adjacent to word edges that leave that state (see the example in Figure 8). Hence, the subset condition cannot hold in word graphs initially generated from acoustic lattices. However, such a word graph can be generated by a series of two-node compression steps; hence, although condition three is uncommon, it is used in our algorithm.

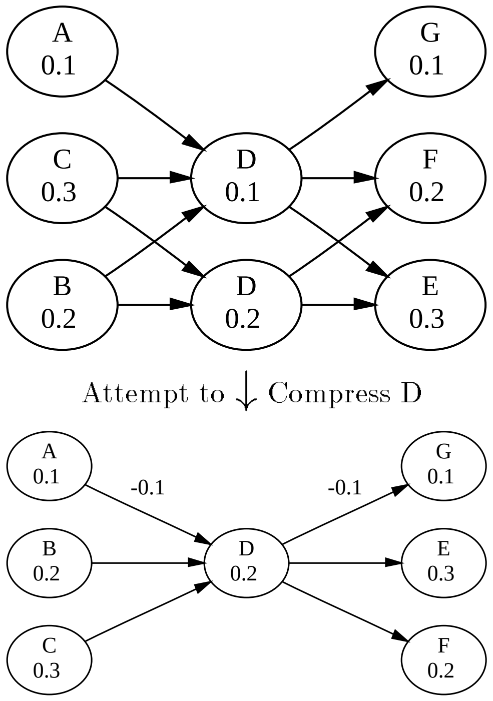


Figure 9: The D nodes cannot be compressed without changing path weights.

A maximally compressed word graph cannot always be produced by two-node compression. For example, consider the word graph in Figure 10 ignoring node and edge weights (weights are set to 0). Clearly the three W nodes are compressible into a single node based on path equivalence; however, they cannot be compressed using two-node compression. Although the word graph in Figure 10 would never be directly generated from a lattice, this structure could be created by a sequence of six two-node compressions on a word graph created from a lattice. Hence, the two-node compression algorithm would not be able to achieve optimality in this case.

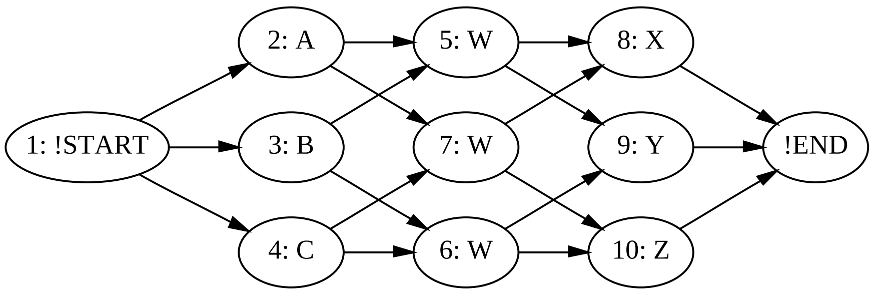


Figure 10: A series of Two-node compressions cannot reduce the size of this word graph although a single node W is sufficient to represent the sentence paths.

Abbreviated pseudocode for two-node compression is shown in Figure 11. When two nodes n_i and n_j meet the third condition then we set v_{new} to be the node with the maximum score and delete the other node. When the first or second condition holds, then n_i and n_j are replaced by a new node v_{new} with SCORE(v_{new}) set to the maximum node's SCORE. The connectivity of v_{new} must also be determined based on PREV(n_i), NEXT(n_i), PREV(n_j), and NEXT(n_j). We can uniformly handle this for condition one and two. PREV(v_{new}) is initially set to PREV(n_i) $\cup$ PREV(n_j), the weights of the tuples are adjusted based on whether MAX is n_i or n_j, and tuples with the same node index are replaced by a single tuple with the maximum score. When a tuple appears in PREV(v_{new}) that was not in PREV(MAX), the weight must be adjusted by subtracting (SCORE(MAX) - SCORE(MIN)) from their prior weight in order to preserve the path score. A similar procedure is used to compute NEXT(v_{new}). At the end of the two-node compression, there is an attempt to move edge weights into the nodes if possible. Weights on nodes and edges can be redistributed arbitrarily, by adding some amount to a node weight and subtracting the same amount from either all of its PREV edges or all of its NEXT edges. Whenever all of the edges incoming to (or outgoing from) a node have the same edge weight, then that weight can be added to the node's score, and the edge weights can then be set to zero. By checking for this condition after each compression step and applying the technique when the condition is met, the algorithm eliminates non-zero edge weights whenever possible, thus maximizing the possibility of additional future compressions. We call this process "pushing" edge weights, since it pushes them from the edges into nodes in the graph.

The running time of the node merging portion of TwoNodeCompression(v_i, v_j) is proportional to the number of edges in the PREV and NEXT sets of v_i and v_j, say E$_{merge}$. Since there is a sorting step, we would be able to carry out the merge process in $O(E_{\text{merge}} \lg E_{\text{merge}})$ time in the worst case. The push step in the code requires time linear in the the number of edges in the PREV and NEXT sets of the compressed node (i.e., $O(E_{\text{merge}})$, as well as in the PREV sets of nodes that follow that node and the NEXT sets of nodes that precede it.

```
TwoNodeCompression($v_i, v_j$){
  MAX-SCORE = max(SCORE($v_i$),SCORE($v_j$))
  // Remember which node has the largest and smallest score
  MAX = argmax(SCORE($v_i$),SCORE($v_j$))
  MIN = argmin(SCORE($v_i$),SCORE($v_j$))
  // Try out each of the conditions and remember the result
  PREV-HOLD = (PREV($v_i$)==PREV($v_j$))
  NEXT-HOLD = (NEXT($v_i$)==NEXT($v_j$))
  SUBSET-HOLD = ((PREV(MIN) $\subseteq_\leq$ PREV(MAX)) $\wedge$ (NEXT(MIN) $\subseteq_\leq$ NEXT(MAX)))
  if SUBSET-HOLD then
    $v_{new}$ = MAX
    Remove MIN from $V$
    Replace all references to MIN with MAX in $G$
  else if PREV-HOLD $\vee$ NEXT-HOLD then {
        Create a new node $v_{new}$
        SCORE($v_{new}$) = MAX-SCORE
        // Determine PREV($v_{new}$) and NEXT($v_{new}$)
        PREV($v_{new}$) = Union(PREV($v_i$),PREV($v_j$))
        NEXT($v_{new}$) = Union(NEXT($v_i$),NEXT($v_j$))
        // Update edge weights to preserve path probabilities
        for each tuple $(v_n, s_n) \in$ NEXT($v_{new}$) do
           if $(v_n, s_n) \notin$ NEXT(MAX) then
              $s_n = s_n$ - (SCORE(MAX) - SCORE(MIN))
        Sort tuples $(v_i, s_i) \in$ NEXT($v_{new}$) in non-decreasing order by node index and then score
        Delete tuples in NEXT($v_{new}$) with the same vertex index as a subsequent tuple
        for each tuple $(v_n, s_n) \in$ PREV($v_{new}$) do
           if $(v_n, s_n) \notin$ PREV(MAX) then
              $s_n = s_n$ - (SCORE(MAX) - SCORE(MIN))
        Sort tuples $(v_i, s_i) \in$ PREV($v_{new}$) in non-decreasing order by node index and then score
        Delete tuples in PREV($v_{new}$) with the same vertex index as a subsequent tuple
        // Update $V$ to reflect the replacement of $v_i$ and $v_j$ by $v_{new}$
        Remove $v_i$ and $v_j$ from $V$
        Add $v_{new}$ to $V$
        Replace all references to $v_i$ and $v_j$ with $v_{new} \in G$ }
  else Return
  // Push edge weights into the previous or following nodes if possible
  for each $(v_n, s_n) \in$ NEXT($v_{new}$) such that $s_n \neq 0$ do {
     if all ($v_i$,SCORE($v_i, v_n$)) $\in$ PREV($v_n$) have the same score then {
       SCORE($v_n$) = SCORE($v_n$) + SCORE($v_i, v_n$)
       for all tuples ($v_i$,SCORE($v_i, v_n$)) $\in$ PREV($v_n$) do
           SCORE($v_i$,$v_n$) = 0 } }
  for each tuple $(v_n, s_n) \in$ PREV($v_{new}$) such that $s_n \neq 0$ do {
     if all ($v_i$,SCORE($v_i, v_n$)) $\in$ NEXT($v_n$) have the same score then {
       SCORE($v_n$) = SCORE($v_n$) + SCORE($v_i, v_n$)
       for all tuples ($v_i$,SCORE($v_i, v_n$)) $\in$ NEXT($v_n$) do
           SCORE($v_i$,$v_n$) = 0 } } }
```

Figure 11: Pseudo-code for two-node compression

Word Graph Compression:

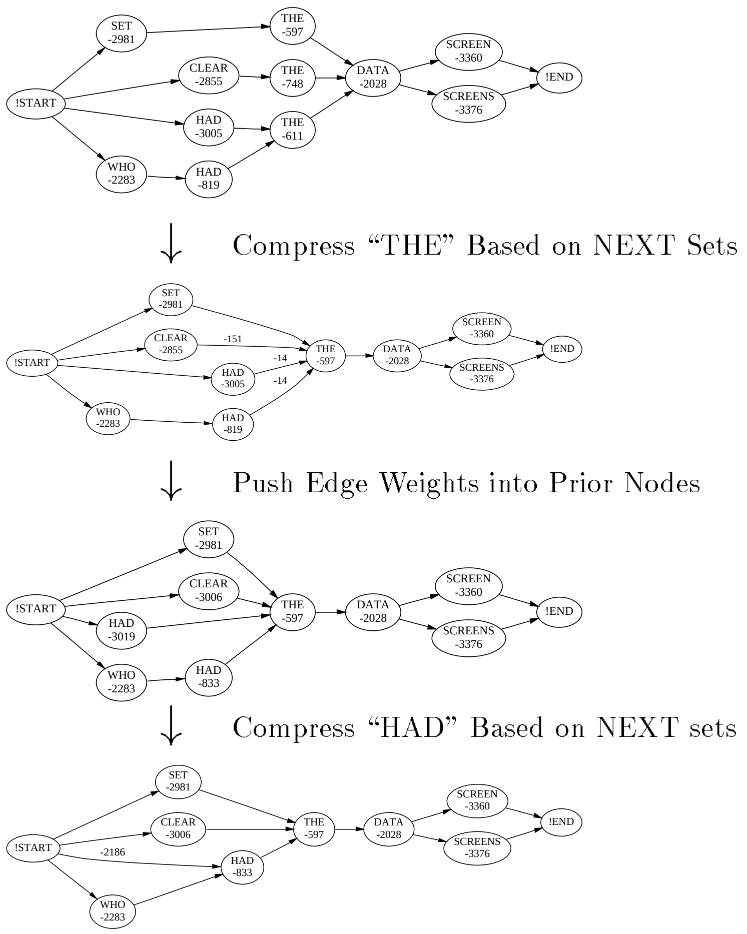


Figure 12: An illustration of word graph compression.

Figure 12 depicts the compression of an example word graph. The compression algorithm is applied to pairs of nodes within the graph, continuing until no further compressions are possible. The ordering of pair-wise compressions can be governed by a number of different strategies. Globally minimizing the number of words in a word graph is difficult; all possible orders of node compression would have to be tried to ensure that the optimal solution is found. In practice, we have found the choice of ordering strategy has only a limited effect on compression results. For experiments, we chose a greedy strategy that works on the largest compressible set of nodes with the same word string using the criterion that affords the greatest compression first. The overall cost of compression is based on the strategy chosen for identifying pairs of nodes to compress. Since each successful two node compression reduces the number of vertices by one, there can be at most $O(V)$ compressions.

4.2 Other Lattice Compression and Pruning Methods

The methods that we compare with the word graph compression algorithm were selected based on their relevance to reducing lattice size and the availability of their implementations.

- Forward-Backward Likelihood Pruning (FBLP)

  The size of lattices can be reduced by pruning paths based on their likelihood, given the fact that a significant proportion of the paths (or arcs) are not likely to be part of the correct sentence. For each arc, the log-likelihood of the best path can be found using an efficient forward-backward algorithm [18, 25]. The forward score for each arc is defined as the overall score of the best sentence hypothesis from the starting frame and ending at time t of the current word arc. Similarly, the backward score for each arc is defined as the overall score of the best partial sentence hypothesis starting from time t of the current word arc until the last frame. Given these values, any arcs that have a best path that is less likely than the most likely hypothesis by some thresholded amount are removed from the lattice. Likelihood pruning can significantly reduce the size of the lattice without decreasing the lattice accuracy or altering the order of the utterance hypotheses remaining in the lattice. It only eliminates the unlikely hypotheses and does not introduce any new paths into the lattice. Forward-backward likelihood pruning differs from pruning during acoustic decoding in that it uses both forward and backward likelihood of each word hypothesis, whereas only the accumulated forward likelihood is used to prune during decoding. The implementation used here is from HTK [2].

- Confusion Networks (CN)

  Mangu's confusion networks approach [12] is another lossy lattice post-processing method. The implementation of this algorithm is available from [11]. To address the discrepancy between the goal of minimizing WER and the metric for maximizing sentence accuracy using Maximum A Posterior (MAP), Mangu developed an algorithm to extract word hypotheses with high posterior probability from the lattices produced by the recognizer. The algorithm groups word hypotheses into time-synchronous slots based on the degree of overlap between time intervals and word phonetic similarity, weighted by the link posterior probability. In the case when the total posterior probability of a group is less than 1, a link "-" representing deletions is added to that position. Figure 13 shows an example of such a clustering produced from the lattice shown in Figure 1.

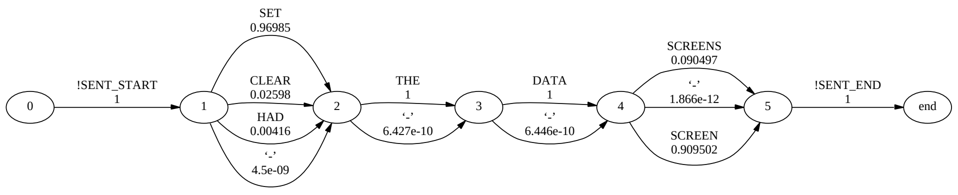


Figure 13: An example of Mangu's confusion network. Each word has a posterior probability associated with it. The symbol "-" on an edge between two nodes represents a NULL linkage.

The algorithm creates a confusion network, which has one node for each equivalence class of the original lattice nodes, and adjacent nodes are connected by one edge per word hypothesis or NULL word "-". Once the network is created, a link pruning step is performed, which deletes links with a likelihood below some threshold with respect to the most likely word hypothesis. Recognition results on the Switchboard task show that an absolute reduction of 1.4% in terms of 1-best WER is obtained after using this post-processing approach compared to a MAP-based baseline system [12].

A confusion network can be thought of as a highly compacted representation of the original lattice in which word hypotheses are ordered based on their posterior probabilities. Note that this algorithm introduces paths that are not in the original lattice. This is usually because a transition is added between two nodes that are not connected in the original lattice. For example, a new path in Figure 13 is "SET THE DATA", which is added due to the existence of the NULL link between node 4 and 5. Mangu found that the newly introduced paths can increase the lattice accuracy and also showed that when confusion networks are used to constrain the search space to run another recognition pass, the newly generated lattices from this run have similar lattice accuracy but smaller size [13].

Unfortunately, confusion networks cannot be directly rescored by another language processing module, so a second recognition pass needs to be run to obtain new lattices for rescoring. In the confusion network, each word hypothesis has a posterior probability, representing the likelihood of uttering this word given the speech signal. This makes the rescoring difficult, since it would be inappropriate to combine this value with a trigram or the other LM score, as is generally done when rescoring the original lattices or the structures produced by the other three post-processing approaches.

- Finite State Machine Determinization and Minimization (FSDM)

  This is a lossless lattice post-processing method. Essentially a word lattice is a non-deterministic finite state machine (FSM) with labels on the transitions indicating a word string and its probability. This type of FSM is called a string-to-weight transducer by Mohri [14], who has devised algorithms for determinizing and minimizing FSM transducers. Mohri et al. [15] have developed a FSM library that supports a variety of operations, including, for example, creation of a FSM, composition of two or more FSMs, determinization of a FSM, and minimization of a FSM. We used this toolkit to determinize and minimize the lattices produced by our speech recognizer.

Mohri [14] extended the sub-set construction method for traditional FSMs to determinize FSM transducers. Due to the properties of lattices, the algorithm runs in $O(|\Sigma|\log|\Sigma| * (|W_1||W_2|)^2))$ time on lattices in the worst case, where $|\Sigma|$ is the vocabulary size, $|W_1|$ is the size of the input lattice, and $|W_2|$ is the size of the output lattice, rather than in exponential time. The determinized transducer maintains the same set of unique paths as found in the original lattice with no alteration of path scores. It is possible for the number of states to increase dramatically due to the process of determinization. For example, a lattice from the 93 WSJ 20K evaluation set increased in size from 993 states to 1600 states, with 14.1% of the lattices from that set increasing in size. Mohri also developed an algorithm for minimizing the states of a determinized FSM transducer that runs in $O(E + Q)$ worst case time on a lattice (which is a directed acyclic graph) where E is the number of edges in the lattice and Q is the number of states. He proved that his state minimization algorithm is optimal in that it produces a FSM with the minimum number of states possible given that it is determinized first. This operation does not eliminate any of the unique paths of the lattice nor affect path scores. The process of determinization and minimization is not guaranteed to reduce the number of states in the lattice. For example, on the 93 WSJ 20K evaluation set, 12.7% of the determinized and minimized lattices increase in size compared to the original.

For applications which need to search the lattice for specific sentence hypotheses, a determinized and minimized lattice 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 time complexity is therefore linear with respect to the number of words in a hypothesis. Similarly, searching for the N-best hypotheses based on path weights is faster. However, if the goal is to apply an algorithm that runs in a time that is a polynomial of the number of words in the lattice, then the word graph compression algorithm that attempts to reduce the number of words in the lattice may be more beneficial to computation time.

4.3 Evaluation of Lattice Post-processing Methods

4.3.1 Experimental Setup

Since we have already conducted pruning experiments using WSJ 92-5K and RM test sets, we will use two additional test sets (RM2 and WSJ 93 test sets) for the following lattice post-processing experiments. The same acoustic model and word pair LM as are used on the RM task are used for RM2. The test set of RM2 consists of four new speakers, each speaking 240 new sentences, for a total of 960 test sentences. For WSJ, the training data is SI284 database (short term speaker independent data from both WSJ0 and WSJ1), containing 284 speakers for a total of about 57 hours of speech. The provided bigram LM is used to generate the recognition network. Both the provided bigram and trigram LM are used to perform forward-backward likelihood pruning. The test set WSJ 93'5K consists of 215 sentences and WSJ 93'20K has 213 sentences. These experimental conditions are chosen in order to evaluate the effectiveness of each method across corpora and different vocabulary sizes.

Lattice post-processing experiments were performed on the lattices produced by our recognizer using acoustic pruning options that were chosen based on the pruning experiments in Section 3, taking into consideration the tradeoff between the performance and the computational cost. Beam width[6] is set to 300, maximum model limit is turned off, word-end beam is set to 200, and the grammar scale is 7 for RM2 and 16 for WSJ.

All the thresholds needed for post-processing lattices were selected based on the corresponding development set using the criterion of reducing lattice size without decreasing accuracy. We chose the threshold to reduce the word density of the representation as much as possible, while decreasing the accuracy as slightly as possible. For RM2, we used RM as the development set, and obtained a threshold of 300 for word pair likelihood pruning and 10^8 for the confusion network approach[7]. For WSJ, we used the 93 development set and obtained a threshold of 250 for bigram and trigram likelihood pruning and 1,200 for the confusion network approach.

We evaluate the four lattice post-processing methods individually and in combination. Forward-backward likelihood pruning preserves the lattice structure and the scores of the remaining hypotheses, and the lattice after likelihood pruning can be further compressed; hence we will evaluate the combination of likelihood pruning and the two lossless compression methods: the word graph compression algorithm and the finite state machine determinization and minimization approach. Because the structure of confusion networks

[6]For RM2, we can use a smaller beam width like 200 or 250 to save computational effort without decreasing accuracy much, but given the fact that recognition speed on RM2 is very fast, we chose to obtain a slight accuracy improvement with little extra computation.

[7]Essentially this is equivalent to turning off pruning. For RM2 we found that for the confusion network approach, only by setting a very large threshold can we obtain accuracy gain while still decreasing the word density.

is not appropriate for lossless compression, we do not combine this approach with either of the compression methods.

4.3.2 Performance Measurement

To evaluate the effectiveness of the lattice post-processing algorithms, we need to measure the complexity and accuracy of the lattices before and after post-processing. Lattice sentence accuracy and lattice word error rate are used to measure accuracy. There are several different methods for measuring lattice complexity. Word density is defined as the number of words in a representation divided by the number of words in the correct sentence. We have employed lattice word density in the previous pruning experiments because it represents the branching factor of a lattice, and thus reflects both lattice size and its complexity. However, if the lattice is to be used in a module that cannot process it as a whole, then it has to find paths and process them in series. In such a case, word density is not an adequate measurement because the number of paths in the representation [3] would need to be considered. Since our goal is to use the lattices (or the compressed word graphs) in a language processing module that processes the input structure in entirety and whose speed is significantly affected by the number of words in the representation, we choose word density as the measurement of both the lattice size and its complexity in the following experiments.

We will also investigate the effect of re-ordering hypotheses by measuring N-best and 1-best recognition results.

4.3.3 Experimental Results

Table 1 and Table 2 show all the post-processing results for the RM2 and WSJ tasks respectively. We measured the average word density and its standard deviation, lattice word error rate, and lattice sentence accuracy of the original lattices and the representations after post-processing lattices. The four methods are abbreviated as: FBLP (forward-backward likelihood pruning), CN (confusion network), FSDM (Finite State determinization and minimization), and WGC (word graph compression). We also indicate the type of LM used by FBLP. For the other methods, the score of the embedded LM during decoding is used.

As can be seen from the tables, when FBLP is performed, the word density is reduced to about half of the size of the original lattices on RM2 with a word pair LM[8], and to one third on the WSJ 5k and 20k sets with a bigram LM. When the trigram LM is used on the WSJ task, the average word density is reduced more compared to the bigram LM, but the standard deviation increases. During trigram likelihood pruning, lattices are first expanded and then pruned based on trigram LM scores. This can possibly increase the lattice size if

[8]To make the experiments parallel to those on the WSJ task, we attempted to construct a bigram LM that would be as effective as the word pair for this task, but the limited sentence set was not sufficient to train a high quality bigram LM. Neither could we build a trigram LM for likelihood pruning.

		RM2		
	Representation	**Density**	**St. Dev**	**GWER (GSAC) (%)**
	original lattices	17.79	12.07	0.38 (97.19)
Individually	word pair-FBLP	8.89	4.27	0.39 (97.19)
	FSDM	14.62	12.06	0.38 (97.19)
	WGC	5.54	2.82	0.38 (97.19)
	CN	2.61	0.57	0.32 (97.6)
In combination	word pair-FBLP + FSDM	7.09	3.76	0.39 (97.19)
	word pair-FBLP + WGC	4.04	1.46	0.39 (97.19)

Table 1: Lattice post-processing results on the RM2 task. A word pair LM is used during decoding to generate the original lattices.

the trigram LM is not strong enough to prune the newly added word edges in the expanded lattice. When FBLP is conducted, lattice word error rate and lattice sentence accuracy are the same as or slightly worse than that of the original lattices. The differences between RM2 and WSJ lattices after pruning is due to the fact that a word pair language model is less effective at reducing the number of words in the lattices and that RM2 is less ambiguous than the WSJ task.

When the original lattices are compressed by FSDM, their average number of words is reduced, however, the standard deviation is close to that of the original lattices on RM2, and more than that of the original lattices on the WSJ task. This is due to the fact that the determinization and minimization of the lattice sometimes increases the lattice size, for example, we found that the size of 12.7% of the lattices on the WSJ 20K set is increased after FSDM is applied. Compared with FBLP, the reduction of lattice word density by FSDM is not so dramatic as by FBLP. Since FSDM is a lossless algorithm, i.e., it does not introduce or remove any hypothesis or alter path scores, the lattice sentence accuracy and lattice word error rate are the same as that of the original lattices.

The word graph compression algorithm is very effective at reducing the lattice size. When the original lattices are compressed by WGC, the number of words is reduced compared to the original size to around 31% on RM2 and 22% on WSJ. The word graph compression algorithm is also lossless and thus does not change the lattice sentence accuracy or word error rate. WGC outperforms FBLP and FSDM on all the tasks, obtaining a lower average word density and standard deviation.

Using the confusion network approach, the lattice word error rate and lattice sentence accuracy are improved compared to the original lattices because of the addition of the new paths to the confusion networks. In fact, this is the only algorithm that increases lattice accuracy among the four post-processing methods. The accuracy increase is more significant on WSJ than RM2 due to the greater difficulty of the WSJ task. The word density of the confusion networks is significantly reduced compared to the original lattices because some

	Representation	WSJ 93-5K		
		Density	**St. Dev**	**GWER (GSAC) (%)**
	original lattices	40.41	12.79	2.16 (73.95)
Individually	2gram-FBLP	13.5	5.24	2.16 (73.95)
	3gram-FBLP	10.56	6.13	2.18 (73.49)
	FSDM	34.01	19.22	2.16 (79.35)
	WGC	8.88	2.24	2.16 (73.95)
	CN	2.96	0.49	1.53 (79.53)
In combination	2gram-FBLP + FSDM	11.21	4.54	2.16 (73.95)
	2gram-FBLP + WGC	5.36	1.39	2.16 (73.95)
	3gram-FBLP + FSDM	6.93	3.90	2.18 (79.49)
	3gram-FBLP + WGC	4.06	1.52	2.18 (73.49)

	Representation	WSJ 93-20K		
		Density	**St. Dev**	**GWER (GSAC) (%)**
	original lattices	45.03	13.27	6.65 (52.11)
Individually	2gram-FBLP	17.96	8.19	6.67 (52.11)
	3gram-FBLP	17.28	11.81	6.76 (52.11)
	FSDM	37.87	15.42	6.65 (52.11)
	WGC	9.68	3.23	6.65 (52.11)
	CN	3.45	0.79	5.37 (54.93)
In combination	2gram-FBLP + FSDM	14.94	6.74	6.67 (52.11)
	2gram-FBLP + WGC	6.4	1.81	6.67 (52.11)
	3gram-FBLP + FSDM	11.35	7.12	6.76 (52.11)
	3gram-FBLP + WGC	5.67	2.4	6.76 (52.11)

Table 2: Lattice post-processing results on the WSJ task. A bigram LM is used during decoding to generate the original lattices.

hypotheses are merged and others are pruned. Among the four individual methods, the confusion network approach yields the smallest word density and standard deviation.

Besides comparing these lattice post-processing methods individually, we also evaluate some of them in combination. When FSDM is applied to the likelihood-pruned lattices, the lattice word density is reduced further compared to using FBLP individually, however, due to the ability of FSDM, the reduction is slight. The lattice word density is still greater than using WGC or CN individually. When WGC is performed on the likelihood-pruned lattices, the number of words is reduced further, to 22% on RM2 and 14% on WSJ (using bigram LM) of the original, with a very slight decrease in lattice accuracy[9]. The combination of

[9]A decrease in accuracy is possible because of likelihood pruning. The word graph compression algorithm does not change the accuracy by itself.

WGC and FBLP is very effective, obtaining smaller lattice word density than all the other methods except CN. Unlike the confusion network approach, which introduces new paths and re-orders paths, the combination of FBLP and WGC only removes the unlikely word hypotheses and does not re-order path hypotheses that remain in the graph.

As described in 4.2, the goal of using a confusion network is to directly minimize WER of the 1-best result, hence, we compare the WER and SAC of the 1-best result of confusion networks both with their original lattices and the lattices after post-processing with the other methods. Results are shown in Table 3. Since likelihood pruning only removes those hypotheses with a very low likelihood without affecting the top hypotheses, and the lossless compression algorithms do not change the rank of any of the hypotheses, FBLP, FSDM and WGC all generate the same 1-best word error rate and sentence accuracy as that of the original lattices after rescoring. When using the confusion network technique, the 1-best results do change. Notice that the 1-best WER is slightly improved on all tasks, although the change of sentence accuracy is not consistent. However, on these tasks, the change of WER is very slight compared to the result reported in [12], where more than 1% absolute decrease of WER was obtained on the Switchboard and Broadcast news tasks. Since the WSJ and RM tasks are less challenging tasks, minimizing WER directly does not generate as much of a gain as on more difficult tasks.

Representation	**WSJ 93-5k**		**WSJ 93-20k**		**RM2**	
	WER(%)	**SAC(%)**	**WER(%)**	**SAC(%)**	**WER(%)**	**SAC(%)**
original lattices, FBLP, FSDM, WGC	9.02	33.02	16.8	23.47	6.34	66.98
CN	8.86	33.02	16.77	23.01	6.28	67.29

Table 3: 1-best recognition results on the RM2 and WSJ tasks.

The confusion network achieves a significant reduction in size by grouping words and pruning unlikely links, but at the same time sentence hypotheses are re-ordered. In order to evaluate the effect of these operations, we also compare the algorithms based on whether the correct sentence appears in the N-best lists extracted from each representation. Note that for the lossless compression algorithms, which do not introduce paths or reorder them, the N-best list will be exactly the same as in the original lattices. Likelihood pruning does not alter the N-best list unless the threshold is set so that some of the hypotheses among the N-best lists are pruned. By contrast, the confusion network algorithm alters path scores, introduces new paths, and can re-order paths in N-best lists. On the WSJ 93 test sets (5K and 20K-vocabulary), we first generate N-best lists from both the confusion networks and the original lattices based on the scores available in each of them. Table 4 shows the number of times that the N-best list contains the correct hypothesis for each data structure as N changes from 1 to 100.

Corpus	N	1	5	10	30	50	100
WSJ 93-5k	original lattices, FBLP, FSDM, WGC	71	122	135	145	150	154
	CN on original lattices	71	120	132	144	147	154
WSJ 93-20k	original lattices, FBLP, FSDM, WGC	50	80	87	98	99	102
	CN on original lattices	49	73	81	88	92	99

Table 4: The number of times that the N-best list extracted from each structure contains the correct sentence. The total number of test sentences in the WSJ 93-5K set is 215, and in the WSJ 93-20K set is 213.

The results in Table 4 show that the confusion network does not have the advantage in terms of N-best list accuracy over the original lattices; much of the time their accuracy is worse than the original lattices. This suggests that although the confusion network has a higher lattice accuracy than the original lattice, i.e., it contains some sentence hypotheses that are not included in the original lattice, it does so at the cost of re-ordering the scores associated with sentence paths such that some of the correct sentences are demoted in their scores. Additionally, the N-best lists extracted from the confusion networks cannot be rescored using more complex language model in a direct way.

Since confusion networks decrease the word error rate of both the lattices and the 1-best recognition results, we will also investigate whether confusion networks can be effectively used to constrain the acoustic recognizer during a second pass as suggested in [13]. The hope is that by constraining the search space using confusion networks, the recognizer avoids some of the previous search errors and promotes the correct hypotheses to be a top candidate. The confusion network constrained recognition results are shown in Table 5. After the second round of acoustic recognition, the size of the newly produced lattices is reduced significantly compared to the original lattices, but the accuracy drops slightly. Although the original confusion network has a higher accuracy than the original lattice and the representations of the other three post-processing methods, the table shows that the feedback framework does not yield an improved recognition result. A merit of this feedback structure is that the new lattices can be employed for rescoring. On the other hand, this approach is far more complicated than the other three post-processing methods and we did not obtain a gain in accuracy by using it.

4.4 Compression Conclusions

The experimental results above demonstrate that the lattices generated by a speech recognizer (with an embedded language model) can be compressed or pruned to further reduce their size significantly.

		1-best results	**Lattice quality**	
Corpus	**Representation**	**WER (SAC) (%)**	**Density**	**GWER (GSAC) (%)**
WSJ 93-5K	original lattices	9.02 (33.02)	35.64	2.16 (73.95)
	CN feedback	9.07 (32.56)	5.12	2.21 (73.95)
WSJ 93-20K	original lattices	16.80 (23.47)	39.12	6.65 (52.11)
	CN feedback	16.98 (23.47)	7.56	6.93 (50.7)

Table 5: 1-best recognition and the new output lattices results when another recognition pass is run using the confusion networks to constrain the search space.

FSDM can make certain that the lattices are deterministic after post-processing, however, the issue of whether the lattices are deterministic is less significant when our goal is to apply a language processing algorithm to the lattices. The new word graph compression algorithm is more effective than the FSDM approach at reducing the lattice word density. By combining the likelihood pruning and the word graph compression, a representation with around 14% of the original number of words (see Table 2) is obtained with only a slight decrease in lattice accuracy on the WSJ task. The effectiveness of the word graph compression algorithm has been verified on different recognition tasks with different vocabulary sizes. Using this post-processing approach, it is possible to loosen pruning parameters during recognition (so long as the computation time is acceptable), and then path prune and compress the word lattices for subsequent processing. This would generate more accurate results than using tight acoustic pruning to create smaller lattices. The word graph compression algorithm also generates word graphs that can be easily rescored by a subsequent language processing module.

The confusion network approach is effective at reducing the word error rate of 1-best result. It also improves the lattice accuracy and reduces the lattice size, but it does so by introducing new paths to the lattice and re-ordering the remaining utterance hypotheses (sometimes demoting the correct hypotheses). Furthermore, the purpose of compression and pruning techniques on lattices is to reduce their size for subsequent processing, but the confusion networks cannot be directly rescored to generate an even more accurate 1-best result. The question of how to directly rescore a confusion network is an open question. Of course one can use the confusion networks to constrain the acoustic recognizer and generate new lattices, but there may be no accuracy gain compared to the original lattices.

Table 6 compares the four post-processing methods on whether they are lossless, whether they can be rescored, their relative size, and their relative accuracy (both lattice accuracy and N-best list accuracy). The up-arrows and down-arrows indicate the direction of change each method has on a particular attribute. The number of arrows for each method under size is used to compare their ability to reduce lattice size (average number of words).

	Lossless	**Rescore**	**Size**	**Lattice accuracy**	**N-best accuracy**
FBLP	no	yes	$\downarrow\downarrow$	possibly$\downarrow$	possibly same*
CN	no	not directly	$\downarrow\downarrow\downarrow\downarrow\downarrow$	mostly$\uparrow$	mostly$\downarrow$
FSDM	yes	yes	$\downarrow$	same	same
WGC	yes	yes	$\downarrow\downarrow\downarrow$	same	same
FBLP + WGC	no	yes	$\downarrow\downarrow\downarrow\downarrow$	possibly $\downarrow$	possibly same*

Table 6: Comparisons of four lattice post-processing methods. '*' indicates that the N-best accuracy is unchanged unless likelihood pruning affects the first N most likely hypotheses.

5 Summary

This paper has reported an investigation of two different factors that affect the quality of word graphs: pruning options during acoustic decoding and post-processing strategies after decoding to reduce lattice size. We systematically studied the effect of pruning during recognition on the quality of the lattices and have found that by combining different pruning options, we can obtain a high quality lattice while moderating computational effort. We have also introduced a new lossless word graph compression algorithm and compared it with several other lattice post-processing algorithms. We found that by combining the likelihood pruning and the word graph compression algorithm, the size of a lattice can be reduced significantly. The time savings achieved by having smaller word graphs can be extremely significant for supporting additional language processing modules.

6 Acknowledgments

This research was supported by Intel, Purdue Research Foundation, and National Science Foundation under Grant No. IRI 97-04358, CDA 96-17388, and BCS-9980054.

References

[1] http://www.ldc.upenn.edu.

[2] http://htk.eng.cam.ac.uk.

[3] Jan W. Amtrup, Henrik Heine, and Uwe Jost. What's in a word graph, evaluation and enhancement of word lattices. Technical report, Universität Hamburg, 1996.

[4] Xavier Aubert and Hermann Ney. Large vocabulary continuous speech recognition using word graph. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 49–52, 1995.

[5] Mary P. Harper and Randall A. Helzerman. Extensions to constraint dependency parsing for spoken language processing. *Computer Speech and Language*, 9:187–234, 1995.

[6] Mary P. Harper, Michael T. Johnson, Leah H. Jamieson, and et al. Interfacing a CDG parser with an HMM word recognizer using word graphs. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 733–736, 1999.

[7] Michael T. Johnson. *Incorporating Prosodic Information and Language Structure Into Speech Recognition Systems.* PhD thesis, Purdue University, 2000.

[8] Michael T. Johnson and Mary P. Harper. Near minimal weighted word graphs for post-processing speech. In *International Workshop on Automatic Speech Recognition and Understanding*, 1999.

[9] Michael T. Johnson, Leah H. Jamieson, and Mary P. Harper. Interfacing acoustic models with natural language processing systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 2419–2422, 1998.

[10] Thomas Kuhn, Pablo Fetter, Alfred Kaltenmeier, and Peter Regel-Brietamann. DP-based wordgraph pruning. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 861–864, 1996.

[11] Lidia L. Mangu. Consensual decosing code. http://nlp.cs.jhu.edu/̃lidia.

[12] Lidia L. Mangu. *Finding Consensus in Speech Recognition.* PhD thesis, Johns Hopkins University, 2000.

[13] Lidia L. Mangu and Eric Brill. Lattice compression in the consensual post-processing framework. In *3rd Conference on Systemics, Cybernetics and Informatics joint with 5th conference on Information systems analysis and synthesis*, pages 246–252, 1999.

[14] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[15] Mehryar Mohri, Fernando C. N. Pereira, and Michael D. Riley. FSM library – general-purpose finite-state machine software tools. http://www.research.att.com/sw/tools/fsm/, 2001.

[16] Hermann Ney, Stefan Ortmanns, and I. Lindam. Extensions to the word graph method for large-vocabulary continuous speech recognition. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 1791–1794, 1996.

[17] Long Nguyen, Richard Schwartz, Ying Zhao, and George Zavaliagkos. Is N-best dead? In *ARPA Human Language Technology Workshop*, pages 386–389, 1994.

[18] Julian J. Odell. *The Use of Context in Large Vocabulary Speech Recognition.* PhD thesis, University of Cambridge, 1995.

[19] Martin Oerder and Hermann Ney. Word graphs: An efficient interface between continuous-speech recognition and language understanding. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 119–122, 1993.

[20] Stefan Ortmanns, Hermann Ney, and Xavier Aubert. A word graph algorithm for large vocabulary continuous speech recognition. *Computer Speech and Language*, pages 43–72, 1997.

[21] P. J. Price, W. Fischer, J. Bernstein, and D. Pallett. A database for continuous speech recognition in a 1000-word domain. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 651–654, 1988.

[22] Fred Richardson, Mari Ostendorf, and Robin Rohlicek. Lattice-based search strategies for large vocabulary speech recognition. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 576–579, 1995.

[23] Richard Schwartz and Yen-Lu Chow. The n-best algorithm: An efficient and exact procedure for finding the N most likely sentence hypotheses. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 81–84, 1990.

[24] Tohru Shimizu, Hirofumi Yamamoto, Hirokazu Masataki, and et al. Spontaneous dialogue speech recognition using cross-word context constrained word graphs. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 145–148, 1996.

[25] Achim Siztus and Stefan Ortmanns. High quality word graphs using forward-backward pruning. In *Proceedings of the IEEE Conference on Acoustic, Speech and Signal Processing*, pages 593–596, 1999.

[26] Wen Wang, Yang Liu, and Mary P. Harper. Rescoring effectiveness of language models using different levels of knowledge and their integration. In *Proceedings of IEEE conference on Acoustic, Speech and Signal Processing*, 2002, To appear.

[27] Fuliang Weng, Andreas Stolcke, and Ananth Sankar. Efficient lattice representation and generation. In *Proceedings of the International Conference on Spoken Language Processing*, pages 2531–2534, 1998.

[28] Steven J. Young, N. H. Russell, and J. H. S. Thornton. Token passing: A simple conceptual model for connected speech recognition systems. Technical report, Cambridge University Engineering Department, 1989.

[29] Steven J. Young, Phil C. Woodland, and Julian J. Odell. Tree-based tying for high accuracy acoustic modelling. In *ARPA Workshop on Human Language Technology*, pages 286–291, 1994.