Network Optimizations for Large Vocabulary Speech Recognition

Mehryar Mohri and Michael Riley

{mohri,riley}@research.att.com
AT&T Labs – Research
180 Park Avenue, Florham Park, NJ 07932-0971

Abstract

The redundancy and the size of networks in large-vocabulary speech recognition systems can have a critical effect on their overall performance. We describe the use of two new algorithms: weighted determinization and minimization [12]. These algorithms transform recognition labeled networks into equivalent ones that require much less time and space in large-vocabulary speech recognition. They are both optimal: weighted determinization eliminates the number of alternatives at each state to the minimum, and weighted minimization reduces the size of deterministic networks to the smallest possible number of states and transitions. These algorithms generalize classical automata determinization and minimization to deal properly with the probabilities of alternative hypotheses and with the relationships between units (distributions, phones, words) at different levels in the recognition system. We illustrate their use in several applications, and report the results of our experiments.

Key words: Large vocabulary speech recognition; search; network optimization; weighted finite-state transducers; stochastic automata.

1 Introduction

The labeled networks used in the search stage of speech recognition systems are often highly redundant. Many paths correspond to the same word contents (word lattices and language models), or to the same phonemes (pronunciation dictionaries) for instance, with distinct weights or probabilities. More generally, at a given state of a network there might be several thousand alternative outgoing arcs, many of them with the same input label. This nondeterminism directly affects the speed of large vocabulary speech recognition systems. *Weighted determinization* allows one to address exactly this problem by reducing the alternatives at each state to the minimum. In other words, the *deterministic* result of the algorithm contains at each state at most one outgoing arc labeled with a given element of the alphabet considered (words, phonemes, etc.).

Other related work has been done to reduce that redundancy using deterministic trees in particular *lexical trees* or the so called *tree-based* representations [3,6,18–20]. The weighted determinization algorithm that we present is a very general algorithm that differs from those approaches by the following: it does not require that networks be *constructed* as trees, it applies to all labeled networks used in speech processing, and it leads to deterministic networks that are in general much more compact than trees. [1] Weighted determinization is not an approximation, a pruning or a heuristic. Its result is exact: for each string, the weight (likelihood) of the best path for that string is the same in the original and in the determinized network.

The size of the deterministic networks used in speech recognition can also affect the overall performance of the systems. We briefly describe an algorithm, *weighted minimization*, for reducing to the minimum the size of deterministic networks.

We describe several applications of weighted determinization and weighted minimization that show the benefits of using these algorithms. They give a significant increase in the time and space efficiency of large vocabulary speech recognition systems by optimizing the networks:

- *word lattices*:
  weighted determinization and weighted minimization help to considerably increase the speed of use of word lattices and to dramatically reduce their size. We report the results of our experiments with word lattices obtained in the DARPA North American Business (NAB) task, and in the DARPA Air Travel Information System (ATIS) task,
- *Context-dependent phone models*:
  context-dependent phone models are very useful in high-accuracy recognition [8,22]. Weighted determinization and minimization can be used to give a very efficient representation of context-dependent models. This is crucial in real-time large-vocabulary speech recognition systems,
- *the NAB task*:
  by reducing considerably the redundancy of networks, weighted determinization leads to a very substantial increase of speed in large vocabulary systems used in the NAB task,
- *the One Million Names task*:
  we present a new task that consists of the real-time discrimination among one million surnames (the One Million Names task). Weighted determiniza-

[1] We use the term *network* here and in some cases in the following instead of *labeled network*, or the more specific terms *acceptors* or *transducers*, to shorten sentences.

tion reduces considerably the number of alternatives in the corresponding networks and weighted minimization their size.

We briefly describe these algorithms, then present their practical use and our experimental results.

2 Algorithms

Weighted determinization and minimization are very general algorithms that apply to weighted acceptors and finite-state transducers. Finite-state transducers are automata in which each transition has an output label in addition to the more familiar input label [5,4]. Weighted acceptors or transducers are acceptors or transducers in which each transition has a weight as well as the input, or input and output labels.

We cannot give a detailed description of these algorithms here. In the following sections, we briefly illustrate the determinization of weighted acceptors, the determinization of weighted transducers, and the minimization of weighted acceptors. We have given elsewhere a full description of these algorithms, including their mathematical basis and proofs of their soundness [9,10,12,13].

2.1 Determinization of Weighted Acceptors

A weighted acceptor or transducer A is said to be *deterministic* [2] iff at each state of A there exists at most one transition labeled with any given element of the input alphabet. Figure 1 gives an example of a non-deterministic weighted acceptor: at state 0, for instance, there are several transitions with the same label a.

We use the same notation for all figures: a bold circle represents an initial state, a double circle a final state. Labels of the transitions are separated from the weights by the symbol '/'. Input and output labels, if any, are separated by ':'.

Weighted determinization applies to a weighted acceptor and outputs an equivalent weighted acceptor that is deterministic. The determinization of weighted acceptors is a generalization of the classical determinization of automata [2]. Unlike the classical case though, not all weighted automata can

[2] The appropriate term used in theoretical computer science is *subsequential* [11].

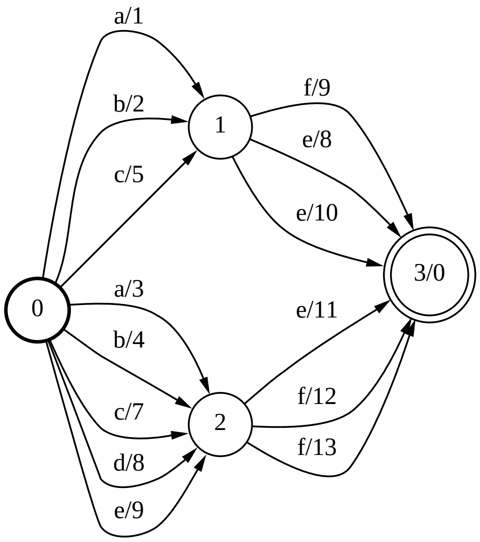

Fig. 1. Non-deterministic weighted automaton A_1.

be determinized. [3] However, most weighted acceptors used in speech processing can be determinized. In particular, any acyclic weighted acceptor admits determinization.

The advantage of weighted determinization is clear: a deterministic acceptor is much more efficient to use than an equivalent non-deterministic one because it allows no alternative at any state. Given an input label a and a state q, there is at most one transition with an input label matching a in the deterministic machines. With non-deterministic weighted acceptors, one needs to explore all the transitions with an input label matching a. For an input string s, these transitions could lead to many different paths that one would need to keep track of in order to compute the weight associated to s. In terms of complexity, the computation of the weight is linear and independent of the size of the machine $|A|$ in the deterministic case ($O(|s|)$), quadratic in the non-deterministic case ($O(|A| \cdot |s|)$). [4] A deterministic weighted automaton is not redundant. It contains at most one path labeled with any given input string. At each state, the choice of the transition to explore is uniquely determined by the input label to match.

[3] We have determined the set of weighted automata that admit determinization and given characterization theorems for *unambiguous* weighted acceptors that admit determinization [12].

[4] We denote by $|s|$ the length of a string s, and by $|A|$ the size of an automaton A.

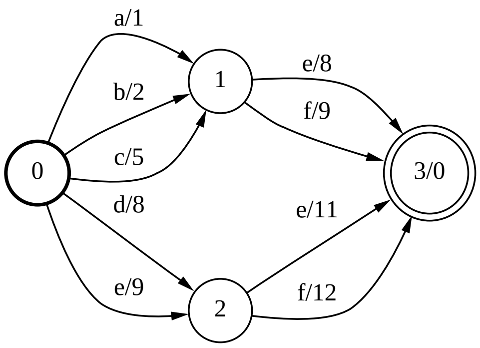

Fig. 2. Equivalent weighted acceptor A_2 obtained by weighted determinization from A_1.

We consider here the common case in speech recognition, where the weights are interpreted as negative logarithms of probabilities. The weight of a path is obtained by adding the weights of its transitions. The output associated to an accepted input string is the minimum of the weights of all paths corresponding to that string. The case where weights are interpreted as probabilities can be treated similarly.

Figure 2 gives the result of weighted determinization for the input acceptor A_1. The two acceptors A_1 and A_2 realize exactly the same function: they associate the same output weight to each input string. As an example, there are two paths corresponding to the input string ae in A_1. The corresponding weights are: $\{1 + 8 = 9, 3 + 11 = 14\}$. The minimum 9 is also the output associated by A_2 to the string ae. This is a general characteristic of determinization: the resultant weighted acceptor is exactly equivalent to the input.

Note that since the two weighted acceptors are equivalent they also have exactly the same n-best list. The list of the best four strings and their weights is identical for both acceptors A_1 and A_2 for instance:

$$(ae, 9), (af, 10), (be, 10), (bf, 11).$$

In general, several successive paths in the enumeration of the best paths might be labeled with the same string if the acceptor is non-deterministic. But for a deterministic acceptor, each path is guaranteed to be labeled with a distinct string. This shows the usefulness of weighted determinization for the computation of n-best lists [12].

The algorithm is close to the classical powerset construction for unweighted automata. [5] However, since transitions with the same input label can have different weights, one can only output the minimum of these weights and needs to keep track of the leftover weight. Therefore, the states of the output of weighted determinization can be viewed as subsets of the set of pairs (q, w), where q is a state of the input acceptor and w the leftover weight.

The initial subset is $\{(i, 0)\}$, where i is the initial state of the input acceptor. For example, for the acceptor A_1 the initial subset is $\{(0, 0)\}$. Each new subset S is processed in turn. For each element of the alphabet a labeling at least one transition leaving a state of S, a new transition t leaving S is constructed in the output machine. The label of t is a and its weight is the minimum of the sums $l + w$ where w is the weight of an a-transition leaving a state p in S and l is p's leftover weight. The destination state of t is the subset S' made of pairs (q, w), where q is a state reached by a transition labeled with a from a state of S, and w is the appropriate leftover weight.

As an example, the state 0 in A_2 corresponds to the initial subset $\{(0, 0)\}$ constructed by the algorithm. The transition leaving 0 in A_2 labeled with a is obtained by considering the two transitions labeled with a leaving the state 0 in A_1. Its weight is obtained by taking the minimum of the weight of these two transitions. Its destination state is the subset $S' = \{(1, -1 + 1 = 0), (2, -1 + 3 = 2)\}$ numbered 1 in A_2.

Note that the order of expansion of the output acceptor does not affect the result. Further, the work done for a given subset S depends only on the elements of S and not on the previous or future work for other subsets. This is an important feature of weighted determinization because it makes it possible to give an on-the-fly implementation of the algorithm. Only states and transitions required by the search algorithm are expanded for a given input string. This plays an important role in the implementation of an efficient n-best decoder and in other applications where one does not wish to expand the entire acceptor.

2.2 Determinization of Weighted Transducers

The determinization of weighted acceptors can be generalized to that of weighted transducers. Weighted transducer determinization applies to a weighted transducer and outputs an equivalent weighted transducer that is deterministic. Here again, unlike the classical case, not all weighted transducers can be de-

[5] The powerset construction is based on the idea that each state of the deterministic automaton corresponds to a set of states of the original non-deterministic one [2].

terminized. [6] But most weighted transducers found in speech recognition applications can be determinized. In particular, any acyclic weighted transducer can be determinized.

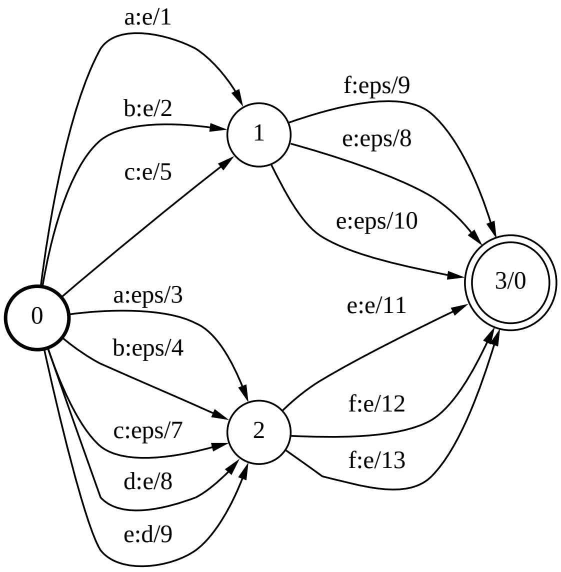

Fig. 3. Non-deterministic weighted transducer T_1.

As in the case of weighted acceptors, the advantage of weighted transducer determinization is to make the transducer more efficient to use by reducing to the minimum the alternatives at each state. Given an input label a and a state q, there is at most one transition with an input label matching a in the deterministic machines. Note that this is not true in general of the output labels.

Figure 3 gives an example of a non-deterministic weighted transducer. T_1 is not deterministic because at state 0 there are several transitions with the same input label a, or same input label b, or c. Similarly, at state 1, there are several transitions with the same input label e, at state 2, several transitions with the same input label f.

T_1 can be determinized. Figure 4 shows the result of determinization applied to T_2. At each state of T_2 the choice of the transition to explore is completely determined by the input label to match.

[6] We have characterized the class of unambiguous weighted transducers that can be determinized [12].

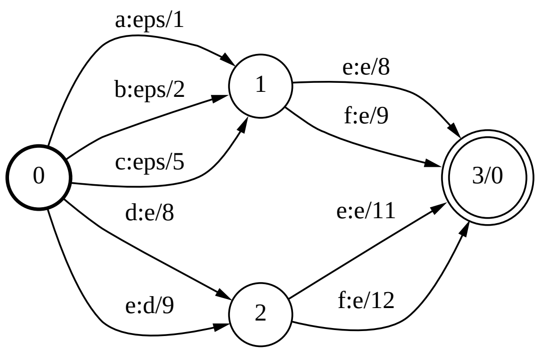

Fig. 4. Determinized weighted transducer T_2.

The two weighted transducers T_1 and T_2 represent exactly the same function. As an example, T_1 admits three paths with the input label be. The output string associated to these paths is the same: $e \cdot \epsilon = \epsilon \cdot e = e$. [7] The weight associated to be by T_1 is obtained by taking the minimum of the weights of these paths: $\min\{2+8, 2+10, 4+11\} = 10$. Thus T_1 associates to be the pair $(e, 10)$. T_2 admits a unique path with input label be. The output label corresponding to that path is also $\epsilon \cdot e = e$, and the weight associated to it: $2 + 8 = 10$.

The algorithm is close to the weighted determinization of acceptors. However, since transitions with the same input label can have different output labels, one can only output the longest common prefix of these outputs and needs to keep track of the leftover strings. Thus here the states of the resultant of weighted determinization correspond to subsets made of tuples (q, s, w), where q is a state of the input transducer, s the leftover string, and w the leftover weight.

The initial subset is $\{(i, \epsilon, 0)\}$, where i is the initial state of the input weighted transducer. For example, for the transducer T_1 the initial subset is $\{(0, \epsilon, 0)\}$. Each new subset S is processed in turn as in the case of acceptors. For each element of the alphabet a labeling the input side of at least one transition leaving a state of S, a new transition t leaving S is constructed in the output machine. The input label of t is a and its weight is the minimum of the sums $l + w$ where w is the weight of an a-transition leaving a state p in S and l is p's leftover weight. The output label is the longest common prefix of the strings $r \cdot s$ where s is the output label of p and r its leftover string.

The destination state of t is the subset S' made of pairs (q, s, w), where q is a state reached by a transition labeled with a from a state of p, s the residual

[7] We denote by ϵ the empty string.

string, and w is the appropriate leftover weight.

As an example, the state 0 in T_2 corresponds to the initial subset $\{(0, \epsilon, 0)\}$ constructed by the algorithm. The transition leaving 0 in T_2 labeled with a is obtained by considering the two transitions with input a leaving the state 0 in T_1. Its output is the longest common prefix of the strings e and ϵ, that is ϵ. Its weight is obtained by taking the minimum of the weight of these two transitions $\min\{1, 3\} = 1$. Its destination state is the subset $S' = \{(1, e, -1+1=0), (2, \epsilon, -1+3=2)\}$ numbered 1 in A_2.

As in the case of acceptors, the order of expansion of the output weighted transducer does not affect the result. The work done for a given subset S depends only on the elements of S and not on the previous or future work for other subsets. Only states and transitions required by the search algorithm are expanded for a given input string. We have given an on-the-fly implementation of weighted transducer determinization.

2.3 Minimization of Weighted Acceptors

Any deterministic finite automaton can be minimized using classical algorithms [1]. In the same way, any deterministic weighted acceptor A can be minimized using our *weighted minimization* algorithm [12].

The resulting weighted acceptor B is equivalent to the acceptor A. It has the minimal number of states and the minimal number of transitions among all equivalent deterministic weighted acceptors equivalent to A.

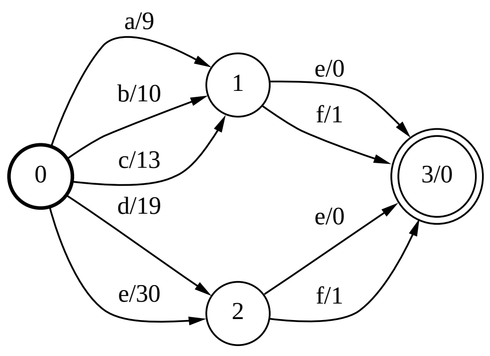

Fig. 5. Equivalent weighted acceptor B_2 obtained by *pushing* from A_2.

Weighted minimization is very efficient. Its time complexity is equivalent to that of the classical minimization, linear in the acyclic case ($O(|Q| + |E|)$), and in $O(|E| \log |Q|)$ in the general case, where Q is the set of states of A and E the set of transitions.

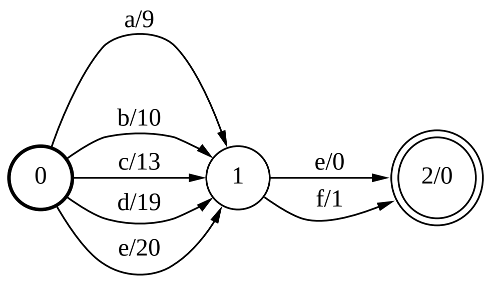

Fig. 6. Equivalent weighted acceptor A_3 obtained by weighted minimization from A_2.

Consider the deterministic weighted acceptor A_2. One can view it as an unweighted acceptor by considering each pair (a, w), made of a label a and a weight w, as a single label, and then apply the classical minimization algorithm to it. But, since the pairs are all distinct, classical minimization would have no effect on the acceptor A_2.

The size of A_2 can still be reduced using (true) weighted minimization. The algorithm works in two steps: the first *pushes* weight along paths, and the second applies the classical minimization algorithm to the result with each distinct label-weight pair viewed as a distinct symbol, as described above.

The pushing step moves the weights of the input acceptor towards the initial state as much as possible. This does not change the topology of the input acceptor and produces an equivalent acceptor. Figure 5 shows the result of pushing for the input A_2. Thanks to pushing, the size of the machine can then be reduced using classical minimization.

Figure 6 illustrates the result of the final step of the algorithm. No approximation or heuristic is used: the resulting acceptor A_3 is equivalent to A_2.

3 Experiments and Results

We have given an efficient implementation of weighted determinization (on-the-fly) of acceptors and transducers, and of weighted minimization. These programs are currently used in speech processing projects at AT&T Labs, at Lucent Bell Laboratories, and at Johns Hopkins University. In the following sections, we describe their use in several speech recognition tasks and report the corresponding results of our experiments. The results show their efficiency and the importance of their use in all these tasks.

3.1 Word Lattices

We applied weighted acceptor determinization to the word lattices obtained in the ARPA ATIS task. This not only made the use of the word lattices more efficient by eliminating their redundancy, but also led to an average reduction of their size by a factor of 9 (table 1).

Table 1
Word lattices in the ATIS task.

	Determinization	Determinization + Minimization
Objects	Reduction factor	Reduction factor
States	≈ 3	≈ 5
Transitions	≈ 9	≈ 17
Paths	$> \mathbf{2^{32}}$	$> \mathbf{2^{32}}$

Figures 7-8 illustrate the weighted determinization in a specific case. Figure 7 corresponds to a word lattice W_1 obtained in the 1,500-word ATIS task. It corresponds to the following utterance: *Show me the flights from Charlotte to Minneapolis on Monday.* Although it is one of the smallest word lattices obtained in this task, W_1 is very complex. It contains more than 151 million paths.

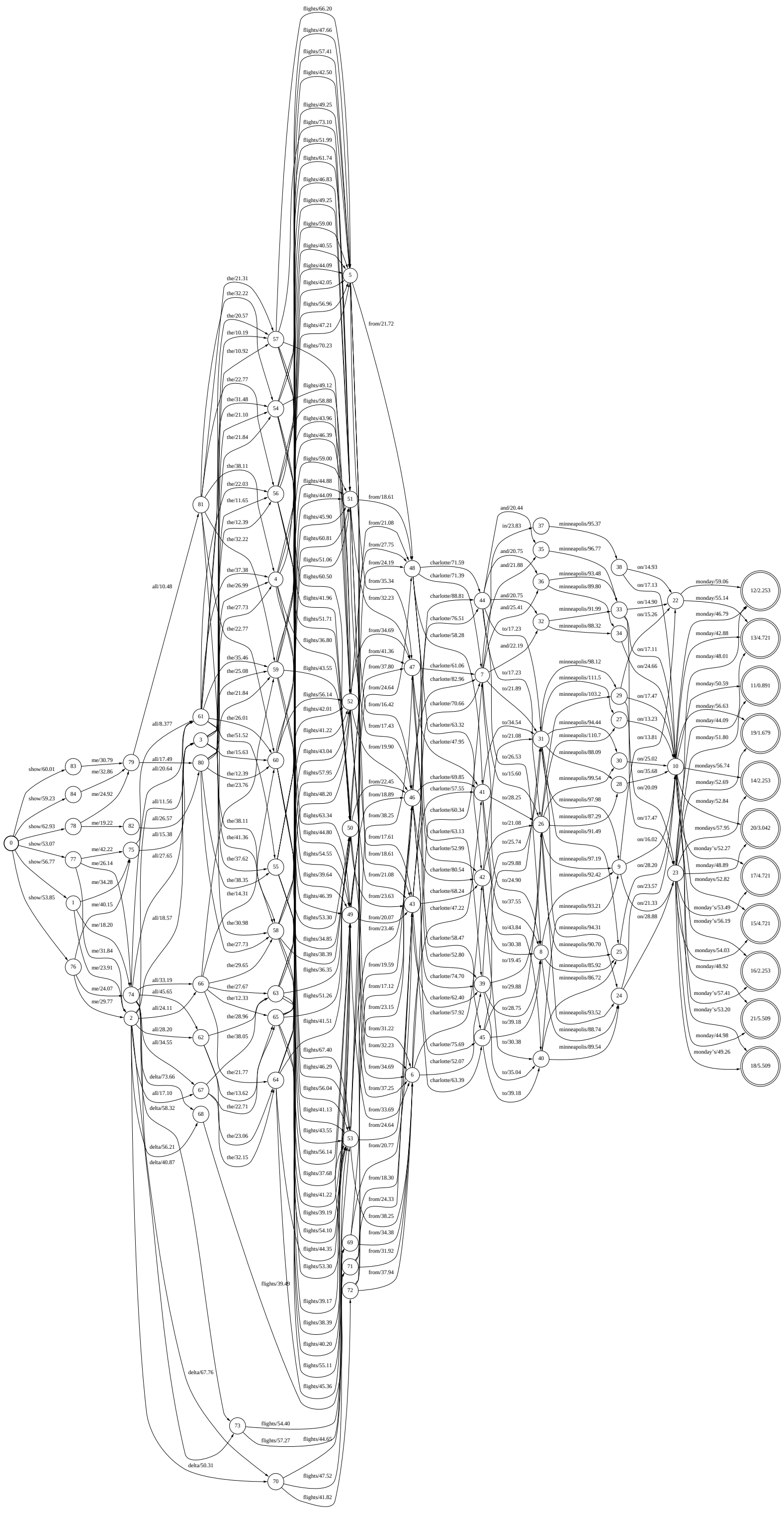

Fig. 7. Word lattice W_1, ATIS task, for the utterance *Show me the flights from Charlotte to Minneapolis on Monday.*

Weighted determinization of acceptors applies to this lattice. This clearly improves the efficiency of the use of the lattice for search or matching purposes. It also reduces the size of the original weighted acceptor by reducing its redundancy to the minimum. The resulting deterministic lattice W_2 contains only 18 paths (figure 8). As mentioned previously, the result is exact: the two lattices realize exactly the same function. The algorithm is very efficient: it took about .06s real time including I/O's (reading and writing the acceptors) to determinize the lattice W_1 on an SGI O2 174 MHz IP32.

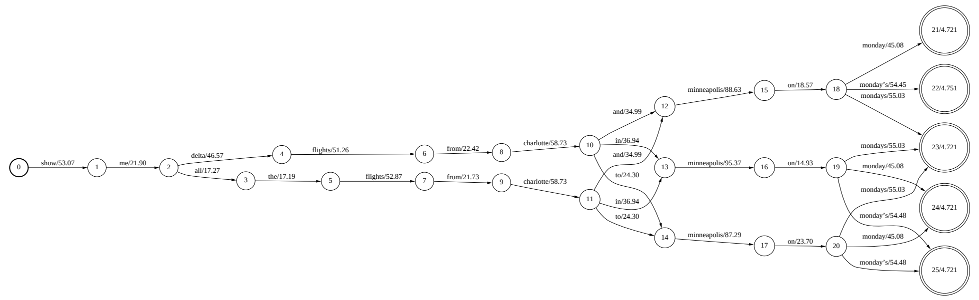

Fig. 8. Equivalent word lattice W_2 obtained by determinization of W_1.

The minimization of weighted acceptors allowed us to reduce further the size of the deterministic weighted automata. The total reduction factor corresponding to the use of determinization and minimization in the ATIS task was about 17 on the average for the word lattices that we used (table 1).

We also applied weighted minimization to deterministic word lattices obtained in the NAB task. On the average, it reduced by a factor of 3 the size of those lattices.

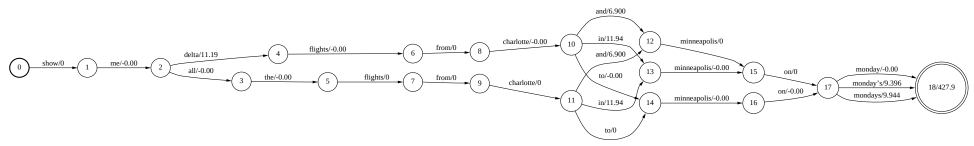

Fig. 9. Equivalent word lattice W_3 obtained by minimization from W_2.

Figure 9 illustrates the use of weighted minimization applied to the deterministic lattice W_2. The resulting machine W_3 has the least number of states and transitions among all equivalent deterministic weighted acceptors. Let us insist that the minimization algorithm is also an exact algorithm: it is not a heuristic or an approximation; the minimal lattice contains exactly the same best paths with exactly the same labels and total weights. The algorithm is very efficient: it took about .07s to minimize the word lattice W_2, including I/O (which take most of this time) on the same computer.

3.2 Context-dependent Phone Models

Context-dependent phone models can be represented by finite-state transducers [21]. They can be built directly in many cases. The transducer of figure 10 for instance can be constructed directly to represent the inverse of a context-dependency model: it maps phone sequences to sequences of names of context-dependent phone models (HMM's). It encodes triphonic context dependency for two hypothetical phones x and y. Each state (a, b) encodes the information that the previous phone was a and the next phone is b; ϵ represents the start or end of a phone sequence and $*$ an unspecified next phone. For instance, it is easy to see that the phone sequence xyx is mapped by the transducer to $x/\epsilon\_y\ y/x\_x\ x/y\_\epsilon$ via the unique state sequence $(\epsilon, *)(x, y)(y, x)(x, \epsilon)$.

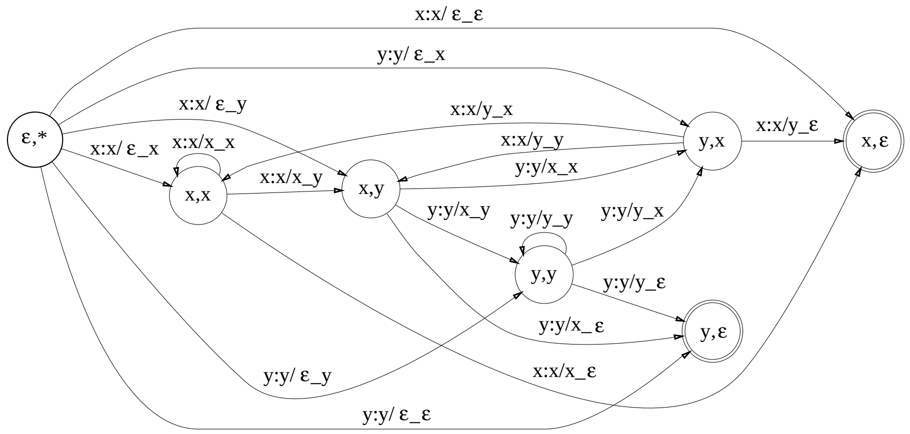

Fig. 10. Non-deterministic context-dependency model.

When the model represents dependencies with contexts of different lengths and different weights, the direct construction might become tedious. One can then use a set of (weighted) rewrite rules to describe the possible contexts. Rewrite rules can be efficiently compiled into finite-state transducers [7,17].

Any context-dependency transducer, whether directly constructed or compiled from patterns or rules, may benefit from transducer determinization.

The transducer of figure 10 is not deterministic. Many states have multiple transitions with the same input label leaving them. The state (x, x), for instance, has three transitions with the same input label x. Transducer determinization can be used to eliminate this redundancy. Figure 11 shows the result of transducer determinization when applied to the transducer of figure 10. [8] At each state of the transducer of figure 11, there is a unique transition labeled with any given phone.

The use of a deterministic context-dependency transducer is important when

[8] \$ is a new end-of-utterance symbol introduced to make the result *sequential*.

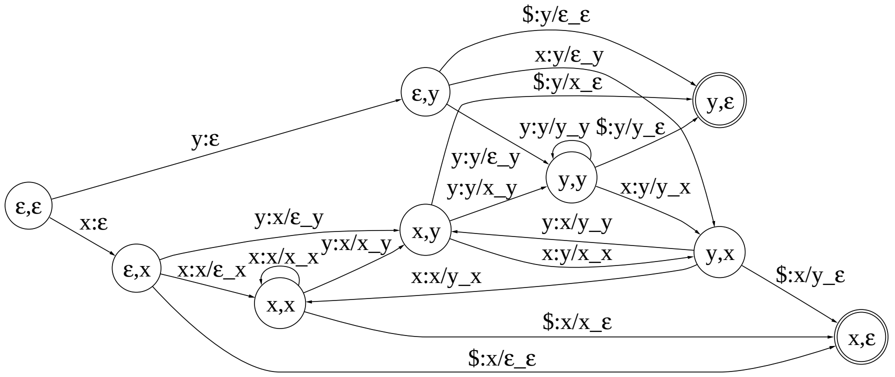

Fig. 11. Deterministic context-dependency model.

combining the different component of a speech recognition system – context-dependency model, pronunciation dictionary, language model. This is because any non-determinism in the context-dependency can increase considerably the size of the resultant

3.3 The NAB Task

In most recognition systems, the words in a language model are (in effect) substituted with their pronunciations to create a large phonemic network. The phonemic network created in this way can have a high degree of nondeterminism in large vocabulary systems. This remains true even when using the efficient finite-state composition techniques that lead to more compact results [14,21], rather than simple substitution.

The phonemic network can contain states with as many (or more) outgoing arcs as the size of the vocabulary. This large number of alternatives can considerably reduce the speed of a recognition system. Determinization of weighted transducers allows one to improve the recognition time by reducing the number of alternatives to the minimum. The number of outgoing arcs in the equivalent deterministic network is at most equal to the number of phones (about 40) versus the size of the vocabulary ($\approx 20,000$) in the original network. [9]

Our experiments in the DARPA North American Business task (NAB) show that weighted determinization plays a crucial role in building a real-time large vocabulary speech recognition system: it reduces recognition time by a factor

[9] Strictly speaking, the phonemic network based on a n-gram model would have an inherent non-determinism due to the backoff model, to fact that a word can be a prefix of another word, and to the presence of homophones. We have solved this by treating ϵs as regular symbols, using word boundary symbols (a distinct word boundary for each homophone), and removing the boundary symbols after determinization.

Table 2
Weighted transducer determinization in the NAB task.

	Reduction factor
Size of transducer	3.3
Recognition time	91.7
Error rate	2.0
Memory used	8.8

of 91.7, reduces the error rate in half, reduces the size of the transducer by a factor of 3.3 and the memory used by a factor of 8.8 (Table 2).

3.4 The One Million Names Task

The determinization of weighted acceptors also allowed us to build a *real-time* system for the recognition of the one million most frequent U.S. surnames found in the Donnelley direct marketing list.

Table 3
Use of determinization and minimization in the One Million Names task.

	Reduction factor
Size of network	5.2
Recognition time	> 100

We used the frequency of the surnames to assign probabilities to the mapping of pronunciations to names. A priori, in some states such as the initial state of the pronunciation network, one million alternatives were possible and the recognition would be very slow. The use of weighted determinization substantially reduced the number of alternatives by limiting it to the number of phones, and led to a real-time recognition system.

We also used weighted minimization after determinization in this task to reduce the size of the networks used by a factor of 5.

The One Million Names task is only an example of the performances that can be achieved in real-time very large vocabulary speech recognition systems for isolated words, using weighted determinization and weighted minimization. We have described elsewhere in great detail a real-time 160,000 word continuous speech recognition system based on the use of weighted transducer determinization [15].

4 Conclusion

The use of weighted determinization and minimization in large vocabulary speech recognition leads to very substantial improvements in performance. In addition, it shows the true degree of redundancy in speech recognition networks, which may not have been fully appreciated previously. In our speech recognition systems we also use another algorithm, *local determinization*, that reduces the redundancy of networks only locally. We report elsewhere on the benefits of local determinization for weighted transducers [16].

The success of these algorithms does not come as a surprise. While most ASR systems use *ad hoc* solutions, thereby limiting the possibility of further improvement and understanding, we believe that general algorithmic solutions based on a sound theoretical foundation can lead to substantially better results.

5 Acknowledgments

We thank Fernando Pereira, Enrico Bocchieri, and Giuseppe Riccardi for discussions, and Andrej Ljolje and Hiyan Alshawi for supplying the ATIS word lattices we used.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison Wesley: Reading, MA, 1974.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques and Tools*. Addison Wesley: Reading, MA, 1986.

[3] G. Antoniol, F. Brugnara, M. Cettolo, and M. Federico. Language model representations for beam-search decoding. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '95)*, pages 588–591, Detroit, MI, 1995.

[4] J. Berstel. *Transductions and Context-Free Languages*. Teubner Studienbucher: Stuttgart, 1979.

[5] S. Eilenberg. *Automata, Languages and Machines*, volume A-B. Academic Press, 1974-1976.

[6] P. S. Gopalakrishnan, L. R. Bahl, and R. L. Mercer. A tree search strategy for large-vocabulary continuous speech recognition. In *Proceedings of the*

International Conference on Acoustics, Speech, and Signal Processing (ICASSP '95), pages 572–575, Detroit, MI, 1995.

[7] R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3), 1994.

[8] K.-F. Lee. Context dependent phonetic hidden Markov models for continuous speech recognition. *IEEE Trans. ASSP*, 38(4):599–609, Apr. 1990.

[9] M. Mohri. Minimization of sequential transducers. *Lecture Notes in Computer Science*, 807, 1994.

[10] M. Mohri. On some applications of finite-state automata theory to natural language processing. *Journal of Natural Language Engineering*, 2:1–20, 1996.

[11] M. Mohri. *Finite-State Language Processing*, chapter On The Use of Sequential Transducers in Natural Language Processing. The MIT Press, 1997.

[12] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:2, 1997.

[13] M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, To appear, 1998.

[14] M. Mohri, F. C. N. Pereira, and M. Riley. Weighted automata in text and speech processing. In *ECAI-96 Workshop, Budapest, Hungary*. ECAI, 1996.

[15] M. Mohri, M. Riley, D. Hindle, A. Ljolje, and F. C. N. Pereira. Full expansion of context-dependent networks in large vocabulary speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, Seattle, Washington, 1998, to appear.

[16] M. Mohri, M. Riley, and R. Sproat. Finite-state transducers in language and speech processing. In *Tutorial at the 16th International Conference on Computational Linguistics (COLING-96), Copenhagen, Denmark*. COLING, 1996.

[17] M. Mohri and R. Sproat. An efficient compiler for weighted rewrite rules. In *34th Meeting of the Association for Computational Linguistics (ACL 96), Proceedings of the Conference, Santa Cruz, California*. ACL, 1996.

[18] J. Odell, V. Valtchev, P. Woodland, and S. Young. A one pass decoder design for large vocabulary recognition. In *Proceedings of the ARPA Human Language Technology Workshop, March 1994*, pages 405–410. Morgan Kaufmann, 1994.

[19] S. Ortmanns, H. Ney, and A. Eiden. Language-model look-ahead for large vocabulary speech recognition. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP'96)*, pages 2095–2098. University of Delaware and Alfred I. duPont Institute, 1996.

[20] S. Ortmanns, H. Ney, F. Seide, and I. Lindam. A comparison of time conditioned and word conditioned search techniques for large vocabulary speech recognition. In *Proceedings of the International Conference on Spoken Language Processing*

(ICSLP'96), pages 2091–2094. University of Delaware and Alfred I. duPont Institute, 1996.

[21] M. Riley, F. C. N. Pereira, and M. Mohri. Transducer composition for context-dependent network expansion. In *Proceedings of Eurospeech'97*. Rhodes, Greece, 1997.

[22] S. Young, J. Odell, and P. Woodland. Tree-based state-tying for high accuracy acoustic modelling. In *ARPA Human Language Technology Workshop*, 1994. Distributed by Morgan Kaufmann, San Francisco.