AN EFFICIENT ALGORITHM FOR THE N-BEST-STRINGS PROBLEM

Mehryar Mohri

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

Michael Riley

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

ABSTRACT

We present an efficient algorithm for solving the n-best-strings problem in a weighted automaton. This problem arises commonly in speech recognition applications when a ranked list of *unique* recognizer hypotheses is desired. We believe this is the first n-best algorithm to remove redundant hypotheses before rather than after the n-best determination. We give a detailed description of the algorithm and demonstrate its correctness. We report experimental results showing its efficiency and practicality even for large n in a $40,000$-word vocabulary North American Business News (NAB) task. In particular, we show that 1000-best generation in this task requires negligible added time over recognizer lattice generation.

1. MOTIVATION

The problem of determining the n shortest paths of a weighted directed graph is a well-studied problem in computer science (see [1] for an extensive bibliography). The problem also admits a number of variants such as finding just the n shortest paths with no cycle, or the n shortest paths with distinct weights, which have all been studied extensively as well. An efficient algorithm introduced by [2] finds an implicit representation of the n shortest paths (allowing cycles and multiple edges) between two nodes in $O(|E| + |Q| \log |Q| + n)$, in constant time per path, after a preprocessing stage dominated by a single-source shortest path computation.

The related problems that arise in speech recognition applications are different. The weighted graphs considered in such applications are weighted automata, typically word or phone lattices representing the alternative hypotheses considered by the recognizer. It is often desirable to determine not just the string labeling a path of the automaton with the lowest total cost, but the n *best distinct strings*, that is the n strings of the automaton with the lowest costs.

Indeed, considering hypotheses other than one corresponding to the best path increases the chances of finding the correct transcription. In many applications, some information source or model not used in the recognizer, such as a more precise grammar or language model, is used to re-rank the n best hypotheses and determine the best candidate. Similarly, when dealing with large tasks, most speech recognition systems use a rescoring method. This consists of first using a simple acoustic and grammar model to produce a word lattice or n-best list, and then to reevaluate these alternative hypotheses with a more sophisticated model.

The automaton searched may contain in general several paths labeled with the same sequence, thus the problem does not coincide with the classical n-shortest-paths problem. In fact, in many applications, the n best paths may contain many times the same sequences. Existing n-shortest-paths algorithms designed for speech recognition applications essentially consist of first determining the k shortest paths with $k >> n$ and then of looking for the n distinct best strings out of the labels of the k paths obtained [3, 4]. This means that a very large number of hypotheses may be generated, compared to previous hypotheses, and then discarded because they are not distinct.

We present an efficient algorithm for solving the n-best-strings problem. The removal of redundant paths is not done as a post-processing step of the n-best search, but prior to it. In this way, we avoid the potentially exponential enumeration of redundant hypotheses. Our method is based on two general algorithms, the determinization of weighted automata [5, 6] and a general n-shortest-paths algorithm [7]. We use weighted determinization to deal with hypothesis redundancy – several paths labeled with the same string – and a single-source shortest paths algorithm to find the n strings with the lowest cost in the result of determinization. An on-demand determinization, augmented to provide the shortest distance information needed in the n-best search, is used so that only that portion of the input visited in the n-best search is determinized.

In practice, we obtain the n-best strings from a speech recognizer by first generating for the entire utterance a lattice that represents the recognizer hypotheses as an acyclic weighted automaton. We then apply the n-best algorithm in a separate step to this automaton. Since lattices with low lattice word error rates can be generated with very little added computation over one-best recognition [8], this approach allows us to completely decouple the n-best generation from the first-pass recognition step with negligible cost in accuracy or efficiency.

We describe our algorithm in detail and report experimental results demonstrating its efficiency. We show that our algorithm is practical even with large n in large-vocabulary speech recognition systems.

2. ALGORITHM

2.1. Preliminaries

The lattices considered in speech recognition applications are typically acyclic weighted automata, however our algorithm is general and applies to all weighted automata. A weighted automaton is a directed weighted graph in which each edge or transition has a label – a phoneme or a word in the case of phone or word lattices [9, 10, 11, 12]. The weights are often interpreted as negative log of probabilities, but in general they may correspond to some other measured quantity. They are added along each path and the weight of a string x is the weight of the minimum weight of a path labeled with x. In what follows, we will thus assume that the automata we are considering are weighted over the semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, called the tropical semiring [13]. However, our results can be straight-forwardedly extended to other cases, in particular to include negative as well as positive real numbers with the semiring $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +, \infty, 0)$.

More formally, a weighted finite automaton (WFA) over the tropical semiring $A = (\Sigma, Q, E, i, F, \lambda, \rho)$ is given by an alphabet or label set Σ, a finite set of states Q, a finite set of *transitions* $E \subseteq Q \times \Sigma \times (\mathbb{R}_+ \cup \{\infty\}) \times Q$, an *initial state* $i \in Q$, a set of *final states* $F \subseteq Q$, an *initial weight* λ and a *final weight function* ρ.

A transition $e = (p[e], l[e], w[e], n[e]) \in E$ can be represented by an arc from the *source* or *previous state* $p[e]$ to the *destination* or *next state* $n[p]$, with the *label* $l[e]$ and *weight* $w[e]$. A path in A is a sequence of consecutive transitions $e_1 \cdots e_n$ with $n[e_i] = p[e_{i+1}]$, $i = 1, \ldots, n-1$. Transitions labeled with the *empty symbol* ϵ consume no input. We denote by $P(R, R')$ the set of paths from a subset of states $R \subseteq Q$ to another subset $R' \subseteq Q$. A *successful path* $\pi = e_1 \cdots e_n$ is a path from the initial state i to a final state $f \in F$. The previous state and next state for path π is the previous state of its initial transition and the next state of its final transition, respectively:

$$p[\pi] = p[e_1], \quad n[\pi] = n[e_n]$$

The label of the path π is the string obtained by concatenating the labels of its constituent transitions:

$$\ell[\pi] = \ell[e_1] \cdots \ell[e_n]$$

The weight associated to path π is the sum of the initial weight (if $p[\pi] = i$), the weights of its constituent transitions:

$$w[\pi] = w[e_1] + \cdots + w[e_n]$$

and the final weight $\rho[n[\pi]]$ if the state reached by π is final. A symbol sequence x is accepted by A if there exists a successful path π labeled with x: $\ell[\pi] = x$. The weight associated by A to the sequence x is then the minimum of the weights of all the successful paths π labeled with x.

2.2. Shortest-distances to final states

The first step of our algorithm consists of computing the shortest distance from each state $q \in Q$ to the set of final states:

$$\phi[q] = \min\{w[\pi] + \rho[f] : \pi \in P(q, f), f \in F\}$$

The distances $\phi[q]$ can be directly computed by running a shortest-paths algorithm from the final states F using the reverse of the digraph. In the case where the automaton contains no negative weights, this can be computed for example using Dijkstra's algorithm in time $O(|E| \log |Q|)$ using classical heaps, or in time $O(|E| + |Q| \log |Q|)$ if we use Fibonacci heaps [7].

After execution of this first step, our algorithm consists of finding the n best paths in the result of an on-the-fly weighted determinization of the automaton A. Thus, we will first give a brief description of weighted determinization – see [5] for a detailed description of the algorithm.

2.3. Determinization and properties

Weighted determinization takes as input a weighted automaton A and outputs an equivalent *subsequential* or *deterministic* automaton B. A weighted automaton B is deterministic if it has a unique initial state and if no two transitions leaving the same state share the same input label. Figure 2 shows the result of the determinization of the automaton in Figure 1.

Unlike the unweighted case, not all weighted automata are determinizable. There exists however a general characterization property and an efficient testing algorithm for checking that property [5, 14]. In particular, any acyclic weighted automaton is determinizable. This guarantees the termination of determinization

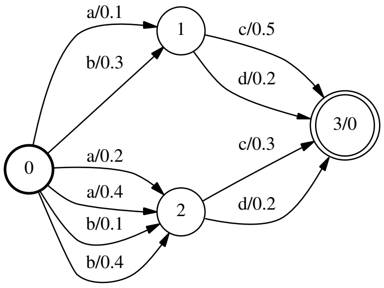

Fig. 1. Non-deterministic weighted automaton A.

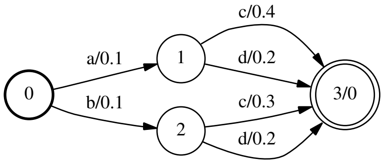

Fig. 2. Equivalent weighted automaton B obtained by determinization of A.

in our case since we are only interested in expanding a finite part of the result of determinization – the part necessary to find the n best paths.

Weighted determinization is a generalization of the classical subset construction [15]. The states of the output automaton correspond to *weighted subsets* $\{(q_0, w_0), \ldots, (q_n, w_n)\}$ where each $q_i \in Q$ is a state of the input machine, and w_i a remainder weight. The algorithm starts with the subset reduced to $\{(i, 0)\}$ and proceeds by creating a transition labeled with $a \in \Sigma$ and weight w leaving $\{(q_0, w_0), \ldots, (q_n, w_n)\}$ if there exists at least one state q_i admitting an outgoing transition labeled with a, w being defined by:

$$w = \min\{w_i + w[e] : e \in E[q_i],\ l[e] = a\}$$

The destination state of that transition corresponds to the subset containing the pairs (q', w') with $q' \in \{n[e] : p[e] = q_i, l[e] = a\}$ and the remainder weight $w' = \min\{w_i + w[e] - w : n[e] = q'\}$. A state is final if it corresponds to a weighted subset S containing a pair (q, w) where q is a final state ($q \in F$) and in that case its final weight is:

$$\min\{w + \rho[q] : (q, w) \in S, q \in F\}$$

An important feature of this algorithm is that it admits a natural on-the-fly implementation. The computation of the transitions leaving a subset S only depends on the states and remainder weights of that subset and on the input automaton, it is independent of the previous subsets visited or constructed. That is we can limit the computation of the result of determinization to just the part that is needed.

Our algorithm precisely utilizes the on-the-fly determinization of the input automaton A. In addition, the shortest-distance information, $\phi[s]$, in A needs to be suitably propagated to the determinized result. For this, we assign an attribute $\Phi(q')$ to each state q' of the result of determinization defined as follows:

$$\Phi[q'] = \min\{w_i + \phi[q_i] : 0 \leq i \leq n\}$$

when q' corresponds to the subset $\{(q_0, w_0), \ldots, (q_n, w_n)\}$. $\Phi[q']$ can be directly computed from each subset constructed. Let $B = (\Sigma, Q', E', i', F', \lambda', \rho')$ be the automaton result of the determinization of A.

Let q' be a state of the result of determinization corresponding to the subset $\{(q_0, w_0), \ldots, (q_n, w_n)\}$. Let $x \in \Sigma^*$ and assume that there exists at least one path labeled with x beginning at a state q_i and leading to F. Define $K(q', x)$ by:

$$K(q', x) = \min_{\pi \in P(q_i, F),\; l[\pi]=x,\; 0 \leq i \leq n} \{w_i + w[\pi] + \rho[n[\pi]]\}$$

If no such path exists, define $K(q', x) = \infty$. The following lemma will be useful in the proof of our main proposition.

Lemma 1 *There exists a unique path π' in B from q' to an element of F' such that $l[\pi'] = x$ and:*

$$K(q', x) = w[\pi'] + \rho'[n[\pi']]$$

Proof. Let π_1 be a path minimizing the quantity defining $K(q', x)$. Without loss of generality, we can assume that $\pi_1 \in P(q_0, F)$. Since q_0 belongs to a subset constructed by determinization, there exists at least one path π_0 from the initial state of A to q_0. Assume π_0 to be such a path with the minimal weight, then by definition of the subset construction, $w[\pi_0] = \alpha + w_0$ where α is the minimum weight of a path labeled with $l[\pi_0]$ leaving the initial state. Thus, $w[\pi_0 \pi_1] = \alpha + w_0 + w[\pi_1] + \rho[n[\pi_1]] = \alpha + K(q', x)$. Since α is a constant independent of q_0, by definition of K, $\pi_0 \pi_1$ is a path of minimal weight in A labeled with $l[\pi_0 \pi_1]$.

By definition of the subset construction, there exists a unique path π'_0 labeled with $l[\pi_0]$ from the initial state to q' in B with weight α and a unique path π'_1 labeled with x from q to F'. Since determinization leads to an equivalent machine, A and B associate the same value to the string $l[\pi_0 \pi_1]$. Thus:

$$w[\pi_0 \pi_1] = \alpha + w[\pi'_1] + \rho'[n[\pi'_1]]$$

And: $K(q', x) = w[\pi'_1] + \rho'[n[\pi'_1]]$. This proves the lemma. □

The following is a property of Φ similar to that of ϕ for A that is crucial for the correctness of our n-best-strings algorithm.

Proposition 1 *For any state $q' \in Q'$ in the determinized automaton B, $\Phi[q']$ represents the shortest distance from q' to the set of final states of B:*

$$\Phi[q'] = \min \{w[\pi'] + \rho'[n[\pi']] : \pi' \in P(q', F')\}$$

Proof. By definition of Φ,

$$\begin{aligned} \Phi[q'] &= \min_{0 \leq i \leq n} \{w_i + \min_{\pi \in P(q_i, F)} \{w[\pi] + \rho[n[\pi]]\}\} \\ &= \min_{0 \leq i \leq n, \pi \in P(q_i, F)} \{w_i + w[\pi] + \rho[n[\pi]]\} \end{aligned}$$

Factoring paths π that share the same label gives:

$$\Phi[q'] = \min_{x \in \Sigma^*} K(q', x)$$

By lemma 1:

$$\Phi[q'] = \min_{x \in \Sigma^*, \pi' \in P(q', F'), l[\pi']=x} w[\pi'] + \rho'[n[\pi']]$$

Since B is deterministic, for each x there exists at most one path leaving q' labeled with x, this expression can thus be simplified and rewritten as:

$$\Phi[q'] = \min_{\pi' \in P(q', F')} w[\pi'] + \rho'[n[\pi']]$$

This ends the proof of the proposition. □

The proposition shows that Φ can be used to determine the shortest distance from each state to F' within B, which we can exploit as in classical A^* single-source shortest-paths algorithms. This is in fact one of the main benefits of the determinization algorithms in this context. This property clearly remains valid regardless of whether the automaton B is completely expanded and constructed or not.

The method we are introducing is very general. Any n-best shortest-paths algorithm taking advantage of Φ can be used to find efficiently the n best paths of B which are exactly labeled with the n best strings of A.

We briefly present here an n-shortest-paths algorithms that is very easy to implement and that we chose to use for our experiments. Other efficient algorithms such as the algorithm of [2] can be used to compute an abbreviated representation of the n best strings as well. [1]

2.4. A simple n-shortest-paths algorithm

The algorithm is a generalization of the classical algorithm of Dijkstra [7]. To simplify the presentation, we will assume that B contains only one final state. This does not affect the generality of our algorithm and the search for the n best paths since one can always complete an automaton by introducing a single final state f to which all previously final states are connected by ϵ-transitions. The following is the pseudocode of the algorithm.

```
1   for p ← 1 to |Q'| do r[p] ← 0
2   π[(i', 0)] ← NIL
3   S ← {(i', 0)}
4   while S ≠ ∅
5         do (p, c) ← head(S); DEQUEUE(S)
6              r[p] ← r[p] + 1
7              if (r[p] = n and p ∈ F) then exit
8              if r[p] ≤ n
9                 then for each e ∈ E[p]
10                          do c' ← c + w[e]
11                               π[(n[e], c')] ← (p, c)
12                               ENQUEUE(S, (n[e], c'))
```

We consider pairs (p, c) of a state $p \in Q'$ and a cost c. The algorithm uses a priority queue S containing the set of pairs (p, c) to examine next. The queue's ordering is based on Φ and defined by:

$$(p, c) < (p', c') \Longleftrightarrow (c + \Phi[p] < c' + \Phi[p'])$$

The algorithm maintains for each state p an attribute $r[p]$ that gives at any time during its execution the number of times a pair (p, c) with first state p has been extracted from S. $r[p]$ is initiated to 0 (line 1) and incremented after each extraction from S (line 6).

Paths are defined by maintaining a predecessor for each pair (p, c) which will constitute a node of the path. The predecessor of the first pair considered, $(i', 0)$, is set to NIL, the beginning of each path (line 2). The priority queue S is initiated to the pair containing the initial state i' of B and the cost 0.

Each time through the loop of lines 4-12 a pair (p, c) is extracted from S (line 5). For each outgoing transition e of p, a new pair $(n[e], c')$ made of the destination state of e and the cost obtained by taking the sum of c and the weight of e is created (lines 9-10). The predecessor of $(n[e], c')$ is defined to be (p, c) and the new pair is inserted in S (lines 11-12).

The algorithm terminates when the n shortest paths have been found, that is when the final state of B has been extracted from S n times (lines 7). Since at most n shortest paths may go through any state p, the search can be limited to at most n extraction of any state p (line 8).

[1] In [16], we also present a simple n-shortest-distance algorithm based on an n-tropical semiring and a generalized shortest-first priority queue.

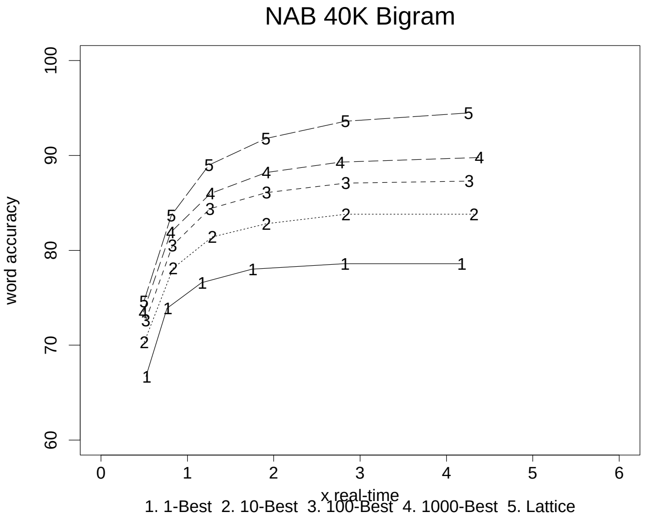

Fig. 3. Experimental results in $40,000$-word vocabulary North American business News (NAB) task.

By construction, in each pair (p, c), c corresponds to the cost of a path from the initial state i' to p and $c + \Phi(p)$ to the cost of that path when completed with a shortest path from p to F.

3. EXPERIMENTS AND RESULTS

We have applied the algorithm described in the previous section to several large-vocabulary speech recognition tasks. Our results show our algorithm to be extremely efficient in practice. Indeed, the additional price to pay to find the n best strings of a lattice is very low even for large n. Here, we report our experiments in the North American business News (NAB) task evaluated on the DARPA Eval '95 test set. We used a 40,000word vocabulary, a bigram language model with 5.8 million n-grams, and a triphonic Gaussian mixture acoustic model with 5000 mixtures and four components per mixture.

Figure 3 shows the accuracy of this recognition system versus real-time when using the n best strings, $n = 1, 10, 100, 1000$, or the entire lattice. The accuracy of an n-best list or lattice here means the word accuracy of the hypothesis that best matches truth among the alternatives returned. The numbers along each curve in the figure correspond to beam widths 9 through 14 in steps of 1. By comparing the same beam widths on curve 1 and curve 5 in Figure 3, we see that this simple NAB system, very similar to the one in [8], produces lattices of good quality in time only a few percent slower than the 1-best result. By comparing the same beam widths on curve 5 and curve 4, we see that the added time for 1000-best generation from the lattice is negligible. In fact, the 1000-best generation experiment, which includes lattice generation, took *less* time than the separate lattice-generation-only experiment at several of the beams, due to small experimental variations in the timing. The experiments were performed on an unloaded 200 MHZ SGI 02000.

4. CONCLUSION

We presented an efficient and general algorithm for solving the n-best-strings problem. To our knowledge, this is the first algorithm designed to solve directly the n-best-strings problem rather than by first determining the k shortest-paths and then finding the n best strings. Our experimental results in large-vocabulary speech recognition tasks demonstrate the efficiency of the algorithm and show that it is practical even with large n. An efficient implementation of this algorithm is incorporated in the FSM library [17] which is available for download for non-commercial use.

5. REFERENCES

[1] David Eppstein, "K shortest paths and other "k best" problems," http://www1.ics.uci.edu/~eppstein/bibs/ kpath.bib, 2001.

[2] David Eppstein, "Finding the k shortest paths," *SIAM J. Computing*, vol. 28, no. 2, pp. 652–673, 1998.

[3] Y. Chow and R Schwartz, "The N-Best Algorithm: An Efficient Procedure for Finding top N Sentence Hypotheses," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '90)*, Albuquerque, New Mexico, April 1990, pp. 81–84.

[4] Frank Soong and Eng-Fong Huang, "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '91)*, Toronto, Canada, November 1991, pp. 705–708.

[5] Mehryar Mohri, "Finite-State Transducers in Language and Speech Processing," *Computational Linguistics*, vol. 23:2, 1997.

[6] Mehryar Mohri and Michael Riley, "Network optimizations for large vocabulary speech recognition," *Speech Communication*, vol. 25:3, 1998.

[7] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, The MIT Press: Cambridge, MA, 1992.

[8] A. Ljolje and F. Pereira and M. Riley, "Efficient general lattice generation and rescoring," in *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary, 1999.

[9] Jean Berstel and Christophe Reutenauer, *Rational Series and Their Languages*, Springer-Verlag: Berlin-New York, 1988.

[10] Arto Salomaa and Matti Soittola, *Automata-Theoretic Aspects of Formal Power Series*, Springer-Verlag: New York, 1978.

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

[12] Samuel Eilenberg, *Automata, Languages and Machines*, vol. A-B, Academic Press, 1974-1976.

[13] Werner Kuich and Arto Salomaa, *Semirings, Automata, Languages*, Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1986.

[14] Cyril Allauzen and Mehryar Mohri, "On the Determinizability of Weighted Automata and Transducers," in *Proceedings of the workshop Weighted Automata: Theory and Applications (WATA)*, Dresden, Germany, March 2002.

[15] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, *Compilers, Principles, Techniques and Tools*, Addison Wesley: Reading, MA, 1986.

[16] Mehryar Mohri, "General Algebraic Frameworks and Algorithms for Shortest-Distance Problems," Technical Memorandum 981210-10TM, AT&T Labs - Research, 62 pages, 1998.

[17] Mohri, Mehryar and Fernando C. N. Pereira and Michael Riley, "General-purpose Finite-State Machine Software Tools," *http://www.research.att.com/sw/tools/fsm*, vol. AT&T Labs – Research, 1997.