Mehryar Mohri
Speech Recognition
Courant Institute of Mathematical Sciences
Homework assignment 1 – Solution
September 24, 2007

A. Genome

[35 points]

Let Σ denote the alphabet $\Sigma = \{A, G, T, C\}$.

1. [5 points] Create a transducer T that implements the edit distance d based on the following insertions, deletions, and substitution costs. For all $a, b \in \Sigma, a \neq b$,

$$\begin{array}{l} d(a,a) = 0, \\ d(a,\epsilon) = d(\epsilon,b) = \frac{1}{2}, \\ d(a,b) = \frac{1}{3}. \end{array} \tag{1}$$

   A simple one-state transducer in the tropical semiring with transitions

$$E = \{(0, x, y, d(x,y), 0) : x, y \in (\Sigma \cup \{\epsilon, a, b\})^2 - \{(\epsilon, \epsilon)\}\}, \tag{2}$$

   can represent this edit-distance, see Figure 1 (for the sake of readability, the figures are given with the alphabet $\Sigma = \{a, b\}$).

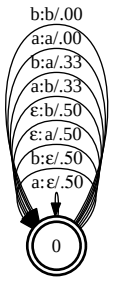

Figure 1: Edit-distance transducer T.

2. [10 points] Using T, find the best alignment between the strings 'AGTCC' and 'GGTACC'. What is the cost of that alignment? What is the complexity of the algorithm you used? Find the second best alignment.

   Simply represent each string with an automaton. Use a shortest-path algorithm, `fsmbestpath`, applied to $X \circ T \circ Y$ obtained via `fsmcompose`, where X and Y are the automata representing these strings.

The result of the composition is acyclic, thus a linear-time shortest-path algorithm can be used. The total complexity is thus $O(|X||T||Y|)$.

3. [10 points] Let A be an automaton accepting the set $X = \{AGTCC, GTACGC\}$ and B an automaton accepting $Y = \{GGTACC, CAGTAC\}$. Using T, A, and B, find the first and second best alignment between the strings in sets X and Y. Give the complexity of your algorithm.

   The same algorithms and operations as in the previous question can be used here with the automata A and B replacing X and Y. The complexity to find the best alignment is $O(|A||T||B|)$.

4. [10 points] Modify the transducer T to take into account the following additional transposition cost: $d(ab, ba) = \frac{1}{4}$. Answer the same as in the previous question.

   The transducer T' of Figure 2 represents this new edit-distance. The same algorithms and operations as in the previous question can be used here with T replaced with T'.

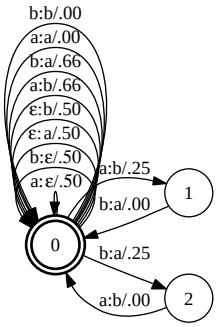

Figure 2: Edit-distance transducer T'.

B. Counting

[35 points]
Let the alphabet be $\Sigma = \{a, b\}$.

1. [10 points] Define a transducer T mapping each input string $x \in \Sigma^*$ to the set of its factors or substrings $\text{Fact}(x) = \{u : x \in \Sigma^* u \Sigma^*\}$.

   It is not hard to see that the transducer T_{fac} of Figure 3 represents this mapping.

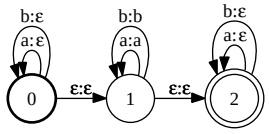

Figure 3: Factor transducer T_{fac}.

2. [15 points] Use T to define a transducer U counting substrings, that is such that $U(x, u)$ be exactly the set of occurrences of the substring u in x for any $x, u \in \Sigma^*$.

   It suffices to augment all transitions and final state of T_{fac} with weight 1 and view the result as a transducer over the $(+, \times)$ semiring, as discussed in class (see Figure 4).

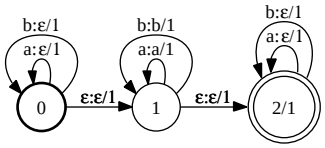

Figure 4: Counting factor transducer T'_{fac}.

3. [10 points] Give an algorithm for finding the number of occurrences of all substrings of length 3 in a document. What is the complexity of your algorithm?

   To do this, one can either define directly a transducer T''_{fac} counting trigrams, or use composition of T'_{fac} with an unweighted transducer extracting trigrams (the unweighted version of T''_{fac}).

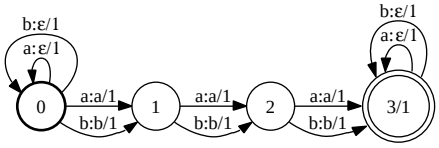

Figure 5: Transducer T''_{fac} counting trigrams.

C. Pronunciation dictionary

[30 points]

1. Consider the following words and their pronunciations (in ARPABET):

any	eh n iy
e.	iy
many	m eh n iy
men	m eh n
per	p er
persons	p er s uh n z
sons	s uh n z
suns	s uh n z
to	t uw
tomb	t uw m
too	t uw
two	t uw

   (a) [5 points] Create a pronunciation lexicon L for these words – i.e., the closure of the pronunciation-to-word transducer.

   It is straightforward to create this transducer using `fsmclosure`.

   (b) [5 points] using L, find all possible word parsings of 't uw m eh n iy p er s uh n z'. Give the result as a graph and as a list of strings in order of fewest to greatest number of words per string.

   This can be done straightforwardly by composing L with an automaton representing this sequence, using `fsmcompose`.

   (c) [20 points] consider the (improbable) bigram language model that gives the cost of word β being followed by word α as:

$$Cost(\alpha|\beta) = |\,\|\alpha\| - \|\beta\|\,|\,,$$

   where $\|\gamma\|$ is the number of phonemes in the pronunciation of the word γ. Create a weighted acceptor that implements this language model. Find the best parsing of the string in (b) when constrained by this language model.

   This can be done by first creating a transducer U mapping each word w to $|w|_{phone}$, the number of phonemes it contains, and composing it with the language model M represented in Figure 6

and projecting on the input, using `fsmcompose` and `fsmproject`. For the sake of readability, here M is restricted to a maximum of 3 phonemes, but the idea of the construction for more phonemes should be clear. This automaton contains exactly one state for each number of phonemes and a transition from state p to q with label q and weight $|p-q|$. Since no word has zero phoneme, there is no need to add the transitions labeled with 0.

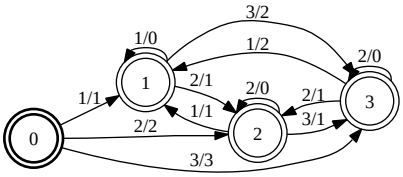

Figure 6: Bigram language model M.