Edit-Distance of Weighted Automata

Mehryar Mohri

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

Abstract. The edit-distance of two strings is the minimal cost of a sequence of symbol insertions, deletions, or substitutions transforming one string into the other. The definition is used in various contexts to give a measure of the difference or similarity between two strings. This definition can be extended to measure the similarity between two sets of strings. In particular, when these sets are represented by automata, their edit-distance can be computed using the general algorithm of composition of weighted transducers combined with a single-source shortest-paths algorithm. More generally, in some applications such as speech recognition and computational biology, the strings may represent a range of alternative hypotheses with associated probabilities. Thus, we introduce the definition of the edit-distance of two distributions of strings given by two weighted automata. We show that general weighted automata algorithms over the appropriate semirings can be used to compute the edit-distance of two weighted automata exactly. The algorithm for computing exactly the edit-distance of weighted automata can be used to improve the word accuracy of automatic speech recognition systems. More generally, the algorithm can be extended to provide an edit-distance automaton useful for rescoring and other post-processing purposes in the context of large-vocabulary speech recognition. In the course of the presentation of our algorithm, we also introduce a new and general *synchronization* algorithm for weighted transducers which, combined with ϵ-removal, can be used to normalize weighted transducers with bounded delays.

1 Introduction

The *edit-distance* of two strings is the minimal cost of a sequence of symbol insertions, deletions, or substitutions transforming one string into the other [17]. It is used in many applications such as computational biology to give a measure of the difference or similarity between two strings.

There exists a classical dynamic-programming algorithm for computing the edit-distance of two strings x_1 and x_2 in time $O(|x_1||x_2|)$, where $|x_i|$, $i = 1, 2$, denotes the length of the string x_i [32]. The algorithm is a special instance of a single-source shortest-paths algorithm applied to a directed graph expanded dynamically. Ukkonen improved that algorithm by reducing the size of the directed graph that needs to be expanded [31]. The complexity of his algorithm is $O(d \max\{|x_1|, |x_2|\})$, where d is the edit-distance of x_1 and x_2. The algorithm is more efficient for strings such that the distance d is small with respect to $|x_1|$ and $|x_2|$. [1]

[1] We refer the reader to [6, 12] for general surveys of edit-distance and other text processing algorithms.

The definition of edit-distance can be extended to measure the similarity between two sets of strings. It is then the minimum of the edit-distance between any two strings, one in each set. In particular, when these sets are represented by (unweighted) automata, their edit-distance can be computed using the general algorithm of composition of weighted transducers combined with a single-source shortest-paths algorithm [23]. We briefly present that algorithm and point out its generality for dealing with more complex edit-distance models including for example transpositions that can be represented by weighted transducers. The classical dynamic programming algorithm for strings can be viewed in fact as a special instance of that algorithm.

In several applications such as speech recognition, computational biology, handwriting recognition, or topic spotting, the strings may represent a range of alternative hypotheses with associated probabilities given as a weighted automaton. Thus, we introduce the definition of the edit-distance of two distributions of strings given by weighted automata, and give a general algorithm for computing that edit-distance.

Note that the number of hypotheses compactly represented by such weighted automata can be very large in many applications. In speech recognition applications for example, they may contain more than four billion distinct strings. Thus, the use of the classical edit-distance computation for strings seems to be prohibitive here: the number of pairs of strings can be larger than four billion squared. The storage and use of the results would also be an issue. Therefore, it is crucial to keep the compact automata representation of the input strings, and provide an algorithm for computing the edit-distance that takes advantage of that representation.

Several heuristics with various approximations have been described to compute that edit-distance in the context of speech recognition applications [18, 11]. We show that general weighted automata algorithms over the appropriate semirings can be used to compute the edit-distance of two weighted automata *exactly*. Our algorithm for computing the edit-distance of weighted automata can be used to improve the word accuracy of automatic speech recognition systems [18]. More generally, the algorithm can be used to provide an *edit-distance automaton* useful for rescoring and other post-processing purposes in the context of large-vocabulary speech recognition.

The paper is organized as follows. We first introduce some preliminary definitions and notation related to semirings and automata that will be used in the rest of the paper (section 2). In section 3, we introduce various definitions related to edit-distances, in particular the definition of the edit-distance of weighted automata. We then give a brief overview of several weighted automata algorithms used to compute the edit-distance of weighted automata (section 4). They include composition of weighted transducers, determinization of weighted automata, ϵ-removal of weighted transducers, and a new and general *synchronization* algorithm for weighted transducers which, combined with ϵ-removal, can be used to normalize weighted transducers with bounded delays. Section 5 presents in detail a general algorithm for computing the edit-distance of two unweighted automata. Finally, the algorithm for computing the edit-distance of weighted automata, the proof of its correctness, and the construction of the edit-distance weighted automaton are given in section 6.

2 Preliminaries

A *weighted automaton* is a finite automaton in which each transition is labeled with some weight element of a *semiring* in addition to the usual alphabet symbol.

Definition 1 ([15]). *A system* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is a* semiring *if:*

1. $(\mathbb{K}, \oplus, \overline{0})$ *is a commutative monoid with identity element* $\overline{0}$*;*
2. $(\mathbb{K}, \otimes, \overline{1})$ *is a monoid with identity element* $\overline{1}$*;*
3. $\otimes$ *distributes over* $\oplus$*;*
4. $\overline{0}$ *is an annihilator for* $\otimes$*: for all* $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$*.*

Thus, a semiring is a ring that may lack negation. Some familiar examples are the Boolean semiring $\mathcal{B} = (\{0,1\}, \vee, \wedge, 0, 1)$, or the real semiring $\mathcal{R} = (\mathbb{R}_+, +, \times, 0, 1)$ used to combine probabilities. Two semirings particularly used in the following sections are: the *log semiring* $\mathcal{L} = (\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$ [20] which is isomorphic to $\mathcal{R}$ via a log morphism with:

$$\forall a, b \in \mathbb{R} \cup \{\infty\}, a \oplus_{\log} b = -\log(\exp(-a) + \exp(-b))$$

where by convention: $\exp(-\infty) = 0$ and $-\log(0) = \infty$, and the *tropical semiring* $\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ which is derived from the log semiring using the Viterbi approximation.

A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *weakly left divisible* if for any x and y in $\mathbb{K}$ such that $x \oplus y \neq \overline{0}$, there exists at least one z such that $x = (x \oplus y) \otimes z$. We can then write: $z = (x \oplus y)^{-1} x$. Furthermore, we will assume then that z can be found in a consistent way, that is: $((u \otimes x) \oplus (u \otimes y))^{-1} (u \otimes x) = (x \oplus y)^{-1} x$ for any $x, y, u \in \mathbb{K}$ such that $u \neq \overline{0}$. A semiring is *zero-sum-free* if for any x and y in $\mathbb{K}$, $x \oplus y = \overline{0}$ implies $x = y = \overline{0}$. Note that the tropical semiring and the log semiring are weakly left divisible since the multiplicative operation, $+$, admits an inverse.

Definition 2. *A* weighted automaton A *over a semiring* $\mathbb{K}$ *is a 7-tuple* $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ *where:*

- Σ *is the finite alphabet of the automaton;*
- Q *is a finite set of states;*
- $I \subseteq Q$ *the set of initial states;*

– $F \subseteq Q$ *the set of final states;*
– $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$ *a finite set of transitions;*
– $\lambda : I \to \mathbb{K}$ *the initial weight function; and*
– $\rho : F \to \mathbb{K}$ *the final weight function mapping F to $\mathbb{K}$.*

Definition 3. *A* weighted finite-state transducer T *over a semiring* $\mathbb{K}$ *is an 8-tuple* $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ *where:*

– Σ *is the finite input alphabet of the transducer;*
– Δ *is the finite output alphabet;*
– Q *is a finite set of states;*
– $I \subseteq Q$ *the set of initial states;*
– $F \subseteq Q$ *the set of final states;*
– $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$ *a finite set of transitions;*
– $\lambda : I \to \mathbb{K}$ *the initial weight function; and*
– $\rho : F \to \mathbb{K}$ *the final weight function mapping F to $\mathbb{K}$.*

Given a transition $e \in E$, we denote by $i[e]$ its input label, $p[e]$ its origin or previous state and $n[e]$ its destination state or next state, $w[e]$ its weight, o[e] its output label (transducer case). Given a state $q \in Q$, we denote by $E[q]$ the set of transitions leaving q.

A *path* $\pi = e_1 \cdots e_k$ is an element of E^* with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. We extend n and p to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$. A cycle π is a path whose origin and destination states coincide: $n[\pi] = p[\pi]$. We denote by $P(q, q')$ the set of paths from q to q' and by $P(q, x, q')$ and $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and output label y (transducer case). These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, R') = \cup_{q \in R,\, q' \in R'} P(q, x, q')$. The labeling functions i (and similarly o) and the weight function w can also be extended to paths by defining the label of a path as the concatenation of the labels of its constituent transitions, and the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $i[\pi] = i[e_1] \cdots i[e_k]$, $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. We also extend w to any finite set of paths Π by setting: $w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi]$. An automaton A is *regulated* if the output weight associated by A to each input string $x \in \Sigma^*$:

$$[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

is well-defined and in $\mathbb{K}$. This condition is always satisfied when A contains no ϵ-cycle since the sum then runs over a finite number of paths. It is also always satisfied with *k-closed* semirings such as the tropical semiring [20]. $[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I, x, F) = \emptyset$.

Similarly, a transducer T is *regulated* if the output weight associated by T to any pair of input-output string (x, y) by:

$$[\![T]\!](x, y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

is well-defined and in $\mathbb{K}$. $[\![T]\!](x) = \emptyset$ when $P(I, x, y, F) = \emptyset$. In the following, we will assume that all the automata and transducers considered are regulated.

A *successful path* in a weighted automaton or transducer M is a path from an initial state to a final state. A state q of M is *accessible* if q can be reached from I. It is *coaccessible* if a final state can be reached from q. A weighted automaton M is *trim* if there is no transition with weight $\overline{0}$ in M and if all states of M are both accessible and coaccessible. M is *unambiguous* if for any string $x \in \Sigma^*$ there is at most one successful path labeled with x. Thus, an unambiguous transducer defines a function.

Note that the second operation of the tropical semiring and the log semiring as well as their identity elements are identical. Thus the weight of a path in an automaton A over the tropical semiring does not change if A is viewed as a weighted automaton over the log semiring or vice-versa.

3 Edit-distance of weighted automata: definitions

Let Σ be a finite alphabet, and let Ω be defined by $\Omega = \Sigma \cup \{\epsilon\} \times \Sigma \cup \{\epsilon\} - \{(\epsilon, \epsilon)\}$. An element ω of the free monoid Ω^* can be viewed as one of $\Sigma^* \times \Sigma^*$ via the concatenation: $\omega = (a_1, b_1) \cdots (a_n, b_n) \to (a_1 \cdots a_n, b_1 \cdots b_n)$. We will denote by h the corresponding morphism from Ω^* to $\Sigma^* \times \Sigma^*$ and write $h(\omega) = (a_1 \cdots a_n, b_1 \cdots b_n)$.

Definition 4. *An alignment ω of two strings x and y over the alphabet Σ is an element of Ω^* such that $h(\omega) = (x, y)$.*

As an example, $(a, \epsilon)(b, \epsilon)(a, b)(\epsilon, b)$ is an alignment of (aba, bb):

$$
\begin{aligned}
x &= a\, b\, a\, \epsilon \\
y &= \epsilon\, \epsilon\, b\, b
\end{aligned}
$$

Let $c : \Omega \to \mathbb{R}_+$ be a function associating some non-negative cost to each element of Ω, that is to each symbol edit operation. For example $c((\epsilon, a))$ can be viewed as the cost of the insertion of the symbol a. Define the cost of $\omega \in \Omega^*$ as the sum of the costs of its constituents: $\omega = \omega_0 \cdots \omega_n \in \Omega^*$: [2]

$$c(\omega) = \sum_{i=0}^{n} c(\omega_i)$$

Definition 5. *The* edit-distance $d(x, y)$ *of two strings x and y over the alphabet Σ is the minimal cost of a sequence of symbol insertions, deletions, or substitutions transforming one string into the other:*

$$d(x, y) = \min \{c(\omega) : h(\omega) = (x, y)\}$$

[2] We are not dealing here with the question of how such weights or costs could be defined. In general, they can be derived from a corpus of alignments using various machine learning techniques such as for example in [27].

In the classical definition of edit-distance, the cost of all edit operations (insertions, deletions, substitutions) is one [17]:

$$\forall a, b \in \Sigma,\ c((a, b)) = 1 \text{ if } (a \neq b)\ ,\ 0 \text{ otherwise}$$

We will denote by c_0 this specific edit cost function. The definition of edit-distance can be generalized to measure the similarity of two sets of strings X and Y by:

$$d(X, Y) = \inf \{d(x, y) : x \in X, y \in Y\}$$

In some applications such as speech recognition or computational biology, one might wish to measure the similarity of a string x with respect to a distribution Y of strings y with probability $P(y)$. The edit-distance of x to Y can then be defined by the expected edit-distance of x to the strings y:

$$d(x, Y) = E_{P(y)}[d(x, y)]$$

Similarly, the edit-distance of two distributions X and Y is defined by:

$$d(X, Y) = E_{P(x,y)}[d(x, y)]$$

We are particularly interested here in the case where these distributions are given by weighted automata which is typical in the applications already mentioned. More precisely, the corresponding automata are acyclic weighted automata over the log semiring.[3] Let A_1 and A_2 be two acyclic weighted automata over the log semiring $\mathcal{L}$ defined over the same alphabet Σ, their edit-distance is thus given by:

$$\begin{aligned} d(A_1, A_2) &= \sum_{x,y} \exp(-[\![A_1]\!](x) - [\![A_2]\!](y)) d(x, y) \\ &= \sum_{x,y} \exp\{-([\![A_1]\!](x) + [\![A_2]\!](y) - \log d(x, y))\} \end{aligned}$$

This equation can be rewritten using the operations of the log semiring.

Definition 6. *The edit-distance of two acyclic weighted automata over the log semiring $\mathcal{L}$, A_1 and A_2, is defined by:*

$$d(A_1, A_2) = \exp(-\bigoplus_{\substack{\log \\ x,y}} ([\![A_1]\!](x) + [\![A_2]\!](y) - \log d(x, y)))$$

We will give an algorithm for computing this edit-distance using general weighted automata algorithms. Note that the computation of $d(A_1, A_2)$ is not trivial a priori since its definition uses operations corresponding to both the tropical semiring (min and $+$ for computing the edit-distance of two strings) and the log semiring ($\oplus_{\log}$ and $+$).

[3] For numerical stability, $-\log$ probabilities are used in practice rather than probabilities.

4 Weighted automata algorithms

In this section we give a brief overview of some classical and existing weighted automata algorithms such as composition, determinization, and minimization, and describe a new and general *synchronization* algorithm for weighted transducers.

4.1 Composition of weighted transducers

Composition is a fundamental operation on weighted transducers that can be used in many applications to create complex weighted transducers from simpler ones. Let $\mathbb{K}$ be a commutative semiring and let T_1 and T_2 be two weighted transducers defined over $\mathbb{K}$ such that the input alphabet of T_2 coincides with the output alphabet of T_1. Then, the composition of T_1 and T_2 is a weighted transducer $T_1 \circ T_2$ defined for all x, y by [3, 9, 28, 15]:[4]

$$[\![T_1 \circ T_2]\!](x, y) = \bigoplus_z T_1(x, z) \otimes T_2(z, y)$$

There exists a general and efficient composition algorithm for weighted transducers [26, 22]. States in the composition $T_1 \circ T_2$ of two weighted transducers T_1 and T_2 are identified with pairs of a state of T_1 and a state of T_2. Leaving aside transitions with ϵ inputs or outputs, the following rule specifies how to compute a transition of $T_1 \circ T_2$ from appropriate transitions of T_1 and T_2:[5]

$$(q_1, a, b, w_1, q_2) \quad \text{and} \quad (q_1', b, c, w_2, q_2') \Longrightarrow ((q_1, q_2), a, c, w_1 \otimes w_2, (q_1', q_2'))$$

In the worst case, all transitions of T_1 leaving a state q_1 match all those of T_2 leaving state q_1', thus the space and time complexity of composition is quadratic: $O((|Q_1| + |E_1|)(|Q_2| + |E_2|))$. Figures 1 (a)-(c) illustrate the algorithm when applied to the transducers of figures 1 (a)-(b) defined over the tropical semiring $\mathcal{T} = T$.

Intersection of weighted automata and composition of finite-state transducers are both special cases of composition of weighted transducers. Intersection corresponds to the case where input and output labels of transitions are identical and composition of unweighted transducers is obtained simply by omitting the weights. Thus, we can use both the notation $A = A_1 \cap A_2$ or $A_1 \circ A_2$ for the intersection of two weighted automata A_1 and A_2. A string x is recognized by A iff it is recognized by both A_1 and A_2 and $[\![A]\!](x) = [\![A_1]\!](x) \otimes [\![A_2]\!](x)$.

[4] Note that we use a *matrix notation* for the definition of composition as opposed to a *functional notation*. This is a deliberate choice motivated by an improved readability in many applications.

[5] See [26, 22] for a detailed presentation of the algorithm including the use of a transducer filter for dealing with ϵ-multiplicity in the case of non-idempotent semirings.

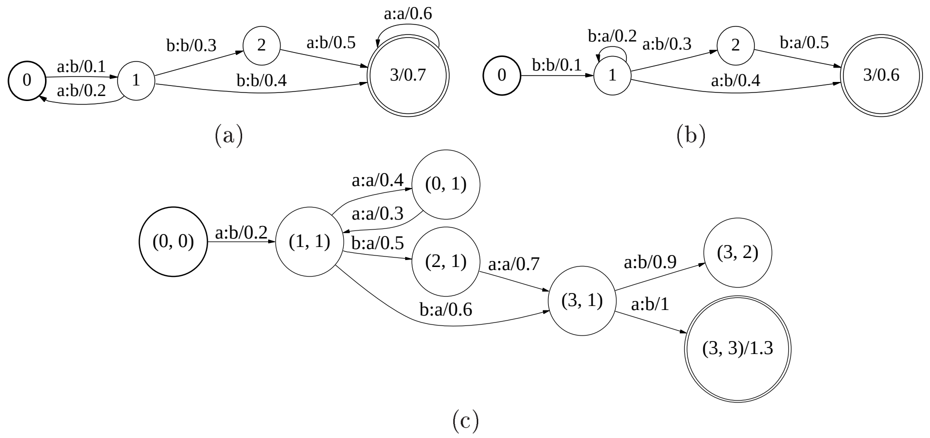

Fig. 1. (a) Weighted transducer T_1 over the tropical semiring. (b) Weighted transducer T_2 over the tropical semiring. (c) Construction of the result of composition $T_1 \circ T_2$. Initial states are represented by bold circles, final states by double circles. Inside each circle, the first number indicates the state number, the second, at final states only, the value of the final weight function ρ at that state. Arrows represent transitions and are labeled with symbols followed by their corresponding weight.

4.2 Determinization of weighted automata

A weighted automaton is said to be *deterministic* or *subsequential* [30] if it has a unique initial state and if no two transitions leaving any state share the same input label.

There exists a natural extension of the classical subset construction to the case of weighted automata over a weakly left divisible semiring called *determinization* [19].[6] The algorithm is generic: it works with any weakly left divisible semiring. Figures 2 (a)-(b) illustrate the determinization of a weighted automaton over the tropical semiring. A state r of the output automaton that can be reached from the start state by a path π corresponds to the set of pairs $(q, x) \in Q \times \mathbb{K}$ such that q can be reached from an initial state of the original machine by a path σ with $l[\sigma] = l[\pi]$ and $\lambda(p[\sigma]) \otimes w[\sigma] = \lambda(p[\pi]) \otimes w[\pi] \otimes x$. Thus, x is the *remaining* weight at state q.

Unlike the unweighted case, determinization does not halt for some input weighted automata. In fact, some weighted automata, non *subsequentiable* automata, do not even admit equivalent subsequential machines. We say that a weighted automaton A is *determinizable* if the determinization algorithm halts for the input A. With a determinizable input, the algorithm outputs an equivalent subsequential weighted automaton [19].

There exists a sufficient condition, necessary and sufficient for unambiguous automata, for the determinizability of weighted automata over a

[6] We assume that the weighted automata considered are all such that for any string $x \in \Sigma^*$, $w[P(I, x, Q)] \neq \overline{0}$. This condition is always satisfied with trim machines over the tropical semiring or any zero-sum-free semiring.

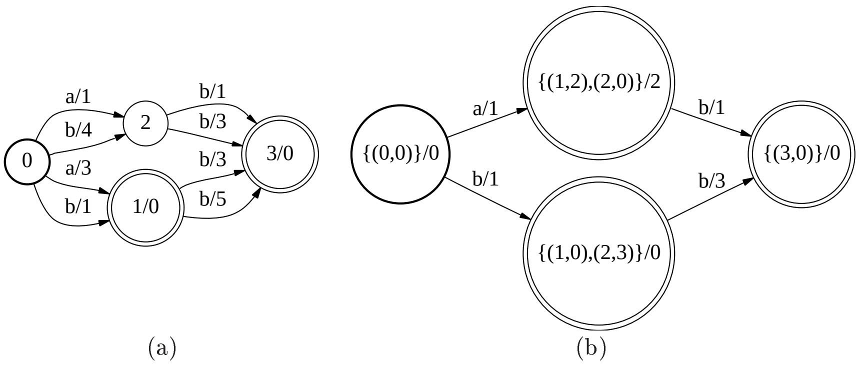

Fig. 2. (a) Weighted automaton A over the tropical semiring. (b) Equivalent subsequential weighted automaton A_2 over the tropical semiring constructed by the determinization algorithm.

commutative semiring based on a *twins property*. There exists an efficient algorithm for testing the twins property for weighted automata [1]. In particular, any acyclic weighted automaton has the twins property and is determinizable.

4.3 Synchronization

In this section, we present a general algorithm for the *synchronization of weighted transducers*. Roughly speaking, the objective of the algorithm is to synchronize the consumption of non-ϵ symbols by the input and output tapes of a transducer as much as possible.

A synchronization procedure was first presented by [9]. The first explicit synchronization algorithm was given by [10] for transducers with bounded delay, later extended by [2] to transducers with constant emission rate. The algorithm of [10] applies only to transducers with input labels different from the empty string ϵ. We present a simple synchronization algorithm that is not restricted to these transducers, and that applies more generally to *weighted* transducers with bounded delays defined over a semiring $\mathbb{K}$.

Definition 7. *The* delay *of a path π is defined as the difference of length between its output and input labels:*

$$d[\pi] = |o[\pi]| - |i[\pi]|$$

The delay of a path is thus simply the sum of the delays of its constituent transitions. A trim transducer T is said to have *bounded delays* if the delay along all paths of T is bounded. We then denote by $d[T] \geq 0$ the maximum delay in absolute value of a path in T. The following lemma gives a straightforward characterization of transducers with bounded delays.

Lemma 1. *A transducer T has bounded delays iff the delay of any cycle in T is zero.*

Proof. If T admits a cycle π with non-zero delay, then $d[T] \geq |d[\pi^n]| = n|d[\pi]|$ is not bounded. Conversely, if all cycles have zero delay, then the maximum delay in T is that of the simple paths which are of finite number. □

We define the *string delay* of a path π as the string $s[\pi]$ defined by:

$$\begin{aligned} s[\pi] &= \text{suffix of } o[\pi] \text{ of length } |d[\pi]| \quad && \text{if } d[\pi] \geq 0 \\ &= \text{suffix of } i[\pi] \text{ of length } |d[\pi]| \quad && \text{otherwise} \end{aligned}$$

and for any state $q \in Q$, the *string delay at q* $S[q]$ by the set of string delays of the paths from an initial state to q:

$$s[q] = \{s[\pi] : \pi \in P(I, q)\}$$

Lemma 2. *If T has bounded delays then the set $s[q]$ is finite for any $q \in Q$.*

Proof. The lemma follows immediately the fact that the elements of $s[q]$ are all of length less than $d[T]$. □

A weighted transducer T is said to be *synchronized* if along any successful path of T the delay is zero or varies strictly monotonically. An algorithm that takes as input a transducer T and computes an equivalent synchronized transducer T' is called a *synchronization* algorithm. We present a synchronization algorithm that applies to all weighted transducers with bounded delays. The following is the pseudocode of the algorithm.

```
Synchronization(T)
F' ← Q' ← E' ← ∅
S ← i' ← {(i, ε, ε) : i ∈ I}
while S ≠ ∅
      do p' = (q, x, y) ← head(S); DEQUEUE(S)
         if (q ∈ F and |x| + |y| = 0)
            then F' ← F' ∪ {p'}; ρ'(p') ← ρ(q)
         else if (q ∈ F and |x| + |y| > 0)
                 then q' ← (f, cdr(x), cdr(y))
                      E' ← E' ∪ (p', car(x), car(y), ρ[q], q')
                      if (q' ∉ Q')
                         then Q' ← Q' ∪ {q'}; ENQUEUE(S, q')
         for each e ∈ E[q]
             do if (|x i[e]| > 0 and |y o[e]| > 0)
                   then q' ← (n[e], cdr(x i[e]), cdr(y o[e]))
                        E' ← E' ∪ {(p', car(x i[e]), car(y o[e]), w[e], q')}
                   else q' ← (n[e], x i[e], y o[e])
                        E' ← E' ∪ {(p', ε, ε, w[e], q')}
                   if (q' ∉ Q')
                      then Q' ← Q' ∪ {q'}; ENQUEUE(S, q')
return T'
```

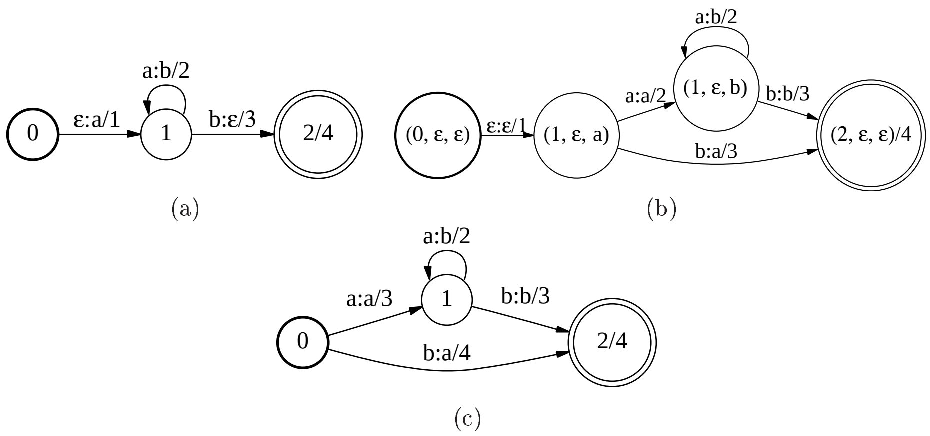

Fig. 3. (a) Weighted automaton A_1 over the tropical semiring. (b) Equivalent synchronized weighted automaton A_2. (c) Synchronized weighted automaton A_3 equivalent to A_1 and A_2 obtained by ϵ-removal from A_2.

To simplify the presentation of the algorithm, we introduce a new state f and add it to Q and F with $\rho[f] = \overline{1}$ and $E[f] = \emptyset$. We denote by $car(x)$ the first symbol of a string x if x is not empty, ϵ otherwise, and denote by $cdr(x)$ the suffix of x such that $x = car(x)\, cdr(x)$.

Each state of the resulting transducer T' corresponds to a triplet (q, x, y) where $q \in Q$ is a state of the original machine T and where $x \in \Sigma^*$ and $y \in \Delta^*$ are strings over the input and output alphabet of T.

The algorithm maintains a queue S that contains at any time the set of states of T' to examine. Each time through the loop of the lines 3-19, a new state $p' = (q, x, y)$ is extracted from S (line 4) and its outgoing transitions are computed and added to E'. The state p' is final iff q is final and $x = y = \epsilon$ and in that case the final weight at p' is simply the final weight at the original state q (lines 5-6). If q is final but the string x and y are not both empty, then the algorithm constructs a sequence of transitions from p' to (f, ϵ, ϵ) to consume the remaining input and output strings x and y (lines 7-11).

For each transition e of q an outgoing transition e' is created for p' with weight $w[e]$. The input and output labels of e' are either both ϵ if $x\,i[e]$ or $y\,o[e]$ is the empty string, the first symbol of these strings otherwise, and the remaining suffixes of these strings are stored in the destination state q' (lines 12-19). Note that in all cases, the transitions created by the steps of the algorithm described in lines 14-17 have zero delay. The state q' is inserted in S if it has never been found before (line 18-19). Figures 3 (a)-(b) illustrate the synchronization algorithm just presented.

Lemma 3. *Let (q, x, y) correspond to a state of T' created by the algorithm. Then, either $x = \epsilon$ or $y = \epsilon$.*

Proof. Let $p' = (q, x, y)$ be a state extracted from S, it is not hard to verify that if $x = \epsilon$ or $y = \epsilon$, then the destination state of a transition leaving p' created by the algorithm is of the form $q' = (r, \epsilon, y')$ or $q' = (r, x', \epsilon)$. Since the algorithm starts with the states (i, ϵ, ϵ), $i \in I$, by induction, for any state $p' = (q, x, y)$ created either $x = \epsilon$ or $y = \epsilon$. $\square$

Lemma 4. *Let π' be a path in T' created by the synchronization algorithm such that $n[\pi']$ corresponds to (q', x', y') with $q' \neq f$. Then, the delay of π' is zero.*

Proof. By construction, the delay of each transition created at lines 14-17 is zero. Since the delay of a path is the sum of the delays of its transitions, this proves the lemma. $\square$

Lemma 5. *Let (q', x', y') correspond to a state of T' created by the algorithm with $q' \neq f$. Then, either $x' \in s[q']$ or $y' \in s[q']$.*

Proof. By induction on the length of π', it is easy to prove that there is a path π' from state (q, x, y) to (q', x', y') iff there is a path from q to q' with input label $x^{-1}i[\pi']x'$, output label $y^{-1}o[\pi']y'$, and weight $w[\pi']$.

Thus, the algorithm constructs a path π' in T' from (i, ϵ, ϵ), $i \in I$, to (q', x', y'), $q' \neq f$ iff there exists a path π in T from i to q' with input label $i[\pi] = i[\pi']x'$, output label $o[\pi] = o[\pi']y'$ and weight $w[\pi] = w[\pi']$. By lemma 4, $|i[\pi']| = |o[\pi']|$. Thus, if $x' = \epsilon$, y' is the string delay of π. Similarly, if $y' = \epsilon$, x' is the string delay of π. By lemma 3, $x' = \epsilon$ or $y' = \epsilon$, thus $y' \in s[q]$ or $x' \in s[q]$. $\square$

Theorem 1. *The synchronization algorithm presented terminates with any input weighted transducer T with bounded delays and produces an equivalent synchronized transducer T'.*

Proof. By lemmas 4 and 5, if (q', x', y') is a state created by the algorithm with $q' \neq f$, then either $x' = \epsilon$ and $y' \in s[q]$ or $y' = \epsilon$ and $x' \in s[q]$. If T has bounded delays, by lemma 2 $s[q]$ is finite, thus the algorithm produces a finite number of states of the form (q', x', y') with $q' \neq f$.

Let (q, x, ϵ) be a state created by the algorithm with $q \in F$ and $|x| > 0$. $x = x_1 \cdots x_n$ is thus a string delay at q. The algorithm constructs a path from (q, x, ϵ) to (f, ϵ, ϵ) with intermediate states $(f, x_i \cdots x_n, \epsilon)$. Since string delays are bounded, at most a finite number of such states are created by the algorithm. A similar result holds for states (q, ϵ, y) with $q \in F$ and $|y| > 0$. Thus, the algorithm produces a finite number of states and terminates if T has bounded delays.

By lemma 4, paths π' in T' with destination state (q, x, y) with $q \neq f$ have zero delay and the delay of a path from a state (f, x, y) to (f, ϵ, ϵ) is strictly monotonic. Thus, the output of the algorithm is a synchronized transducer. This ends the proof of the theorem. $\square$

The algorithm creates a distinct state (q, x, ϵ) or (q, ϵ, y) for each string delay $x, y \in s[q]$ at state $q \neq f$. The paths from a state (q, x, ϵ) or (q, ϵ, y),

$q \in F$, to (f, ϵ, ϵ) are of length $|x|$ or $|y|$. The length of a string delay is bounded by $d[T]$. Thus, there are at most $|\Sigma|^{\leq d[T]} + |\Delta|^{\leq d[T]} = O(|\Sigma|^{d[T]} + |\Delta|^{d[T]})$ distinct string delays at each state. Thus, in the worst case, the size of the resulting transducer T' is:

$$O((|Q| + |E|)(|\Sigma|^{d[T]} + |\Delta|^{d[T]}))$$

The string delays can be represented in a compact and efficient way using a suffix tree. Indeed, let U be a tree representing all the input and output labels of the paths in T found in a depth-first search of T. The size of U is linear in that of T and a suffix tree V of U can be built in time proportional to the number of nodes of U times the size of the alphabet [14], that is in $O((|\Sigma| + |\Delta|) \cdot (|Q| + |E|))$. Since each string delay x is a suffix of a string represented by U, it can be represented by two nodes n_1 and n_2 of V and a position in the string labeling the edge from n_1 to n_2. The operations performed by the algorithm to construct a new transition require either computing xa or $a^{-1}x$ where a is a symbol of the input or output alphabet. Clearly, these operations can be performed in constant time: xa is obtained by going down one position in the suffix tree, and $a^{-1}x$ by using the suffix link at node n_1. Thus, using this representation, the operations performed for the construction of each new transition can be done in constant time. This includes the cost of comparison of a newly created state (q', x', ϵ) with an existing state (q, x, ϵ), since the comparison of the string delays x and x' can be done in constant time. Thus, the worst case space and time complexity of the algorithm is:

$$O((|Q| + |E|)(|\Sigma|^{d[T]} + |\Delta|^{d[T]}))$$

This is not a tight evaluation of the complexity since it is not clear if the worst case previously described can ever occur, but the algorithm can indeed produce an exponentially larger transducer in some cases.

Note that the algorithm does not depend on the queue discipline used for S and that the construction of the transitions leaving a state $p' = (q, x, y)$ of T' only depends on p' and not on the states and transitions previously constructed. Thus, the transitions of T' can be naturally computed on-demand. We have precisely given an on-the-fly implementation of the algorithm and incorporated it in a general-purpose finite-state machine library (FSM library) [23, 25]. Note also that the additive and multiplicative operations of the semiring are not used in the definition of the algorithm. Only $\overline{1}$, the identity element of $\otimes$, was used for the definition of the final weight of f. Thus, to a large extent, the algorithm is independent of the semiring $\mathbb{K}$. In particular, the behavior of the algorithm is identical for two semirings having the same identity elements, such as for example the tropical and log semirings.

4.4 ϵ-Removal

The result of the synchronization algorithm may contain ϵ-transitions (transitions with both input and output empty string) even if the input contains none. An equivalent weighted transducer with no ϵ-transitions can be computed from T' using a general ϵ-removal algorithm [21]. Figure 3 (c) illustrates the result of that algorithm when applied to the synchronized transducer of figure 3 (b).

Since ϵ-removal does not shift input and output labels with respect to each other, the result of its application to T' is also a synchronized transducer.

Note that the synchronization algorithm does not produce any ϵ-cycle if the original machine T does not contain any. Thus, in that case, the computation of the ϵ-closures in T can be done in linear time [21] and the total time complexity of ϵ-removal is $O(|Q'|^2 + (T_\oplus + T_\otimes)|Q'| \cdot |E'|)$, where $T_\oplus$ and $T_\otimes$ denote the cost of the $\oplus$, $\otimes$ operations in the semiring $\mathbb{K}$. Also, on-the-fly synchronization can be combined with on-the-fly ϵ-removal to *directly* create synchronized transducers with no ϵ-transition on-the-fly.

A by-product of the application of synchronization followed by ϵ-removal is that the resulting transducer is *normalized.*

Definition 8. *Let π and π' be two paths of a transducer T with the same input and output labels: $i[\pi] = i[\pi']$ and $o[\pi] = o[\pi']$. We will say that $\pi = e_1 \cdots e_n$ and $\pi' = e'_1 \cdots e'_{n'}$ are* identical *if they have the same number of transitions ($n = n'$) with the same labels: $i[e_k] = i[e'_k]$ and $o[e_k] = o[e'_k]$ for $k = 1, \ldots, n$. T is said to be* normalized *if any two paths π and π' with the same input and output labels are identical.*

Note that the definition does not require the weights of two identical paths to be the same.

Lemma 6. *Let T be a synchronized transducer and assume that T has no ϵ-transition. Then, T is normalized.*

Proof. Let π and π' be two paths with the same input and output labels. Since T is synchronized and has no ϵ-transitions, π and π' have the same delay and more precisely the delay varies in the same way along these two paths and are thus identical. □

5 Computation of the edit-distance of unweighted automata

The edit-distance $d(X, Y)$ of two sets of strings X and Y each represented by an unweighted automaton can be computed using the general algorithm of composition of transducers and a single-source shortest-paths algorithm [23]. The algorithm applies similarly in the case of an arbitrarily complex edit-distance defined by a weighted transducer over the tropical semiring.

Let A_1 and A_2 be two (unweighted) automata representing the sets X and Y. By definition, the edit-distance of X and Y, or equivalently that of A_1 and A_2, is defined by:

$$d(A_1, A_2) = \inf \{d(x, y) : x \in Dom(A_1), y \in Dom(A_2)\}$$

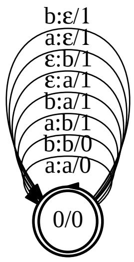

Fig. 4. Weighted transducer over the tropical semiring representing Ψ^* with the edit cost function c_0 and $\Sigma = \{a, b\}$.

5.1 Alignment costs in the tropical semiring

Let Ψ be the formal power series defined over the alphabet Ω and the tropical semiring by: $(\Psi, (a, b)) = c((a, b))$ for $(a, b) \in \Omega$.

Lemma 7. *Let $\omega = (a_0, b_0) \cdots (a_n, b_n) \in \Omega^*$ be an alignment, then (Ψ^*, ω) is exactly the cost of the alignment ω.*

Proof. By definition of the $+$-multiplication of power series in the tropical semiring:

$$\begin{aligned}
(\Psi^*, \omega) &= \min_{u_0 \cdots u_k = \omega} (\Psi, u_0) + \cdots + (\Psi, u_k) \\
&= (\Psi, (a_0, b_0)) + \cdots + (\Psi, (a_n, b_n)) \\
&= \sum_{i=0}^{n} c((a_i, b_i)) = c(\omega)
\end{aligned}$$

This proves the lemma. □

Ψ^* is a rational power series as the closure of the polynomial power series Ψ [28, 4]. Thus, by the theorem of Schützenberger [29], there exists a weighted automaton A defined over the alphabet Ω and the semiring $\mathcal{T}$ realizing Ψ^*. A can also be viewed as a weighted transducer T with input and output alphabets Σ. Figure 5 shows the simple finite-state transducer T realizing Ψ^* in the particular case of the edit cost function c_0 and with $\Sigma = \{a, b\}$.

5.2 Algorithm

By definition of composition of transducers and by lemma 7, the weighted transducer $A_1 \circ T \circ A_2$ contains a successful path corresponding to each alignment ω of a string accepted by A_1 and a string accepted by A_2 and the weight of that path is $c(\omega)$.

Theorem 2. *Let U be the weighted transducer over the tropical semiring obtained by: $U = A_1 \circ T \circ A_2$. Let π be a shortest path of U from the initial state to the final states. Then, π is labeled with one of the best alignments of a string accepted by A_1 and a string accepted by A_2 and: $d(A_1, A_2) = w[\pi]$.*

Proof. The result follows directly the previous remark. □

The theorem provides an algorithm for computing the best alignment between the strings of two unweighted automata A_1 and A_2 and for computing their edit-distance $d(A_1, A_2)$. Any single-source shortest-paths algorithm applied to U can be used to compute the edit-distance and a best alignment. Let $|V|$ denote the sum of the number of states and transitions of an automaton or transducer V. Since the worst case time and space complexity of composition is quadratic, we have: $|U| = O(|A_1||A_2|)$.

When U is acyclic, this is the case in particular when A_1 and A_2 are both acyclic, the total time complexity of the computation of the best alignment and the edit-distance $d(A_1, A_2)$ is:

$$O(|A_1||A_2|)$$

since we can then use Lawler's linear-time single-source shortest paths algorithm [16, 5]. In the general case, the total complexity of the algorithm is:

$$O(|E| + |Q| \log |Q|)$$

where E denotes the set of transitions and Q the set of states of U, using Dijkstra's algorithm implemented with Fibonacci heaps [7, 5].

In particular, the time complexity of the computation of the edit-distance for two strings x and y is $O(|x||y|)$. The classical dynamic programming algorithm for computing the edit-distance of two strings can in fact be viewed as a special instance of the more general algorithm just presented.

This algorithm is very general. It extends to the case of automata the classical edit-distance computation and it also generalizes the classical definition of edit-distance. Indeed, any weighted transducer with non-negative weights can be used here without modifying the algorithm. [7] Edit-distance transducers with arbitrary topologies, arbitrary number of

[7] Transducers with negative weights can be used as well, but the single-source shortest-paths problems of Bellman-Ford would need to be used then in general, which can make the algorithm less efficient in general.

states and transitions can be used instead of the specific one-state edit-distance transducer used in most applications. More general transducers assigning non-negative costs to transpositions or to more general weighted context-dependent rules [24] can be used to model complex edit-distances.

6 Computation of the edit-distance of weighted automata

Our algorithm for computing the edit-distance of two weighted automata is based on weighted composition, determinization, ϵ-removal, and synchronization. More precisely, our algorithm computes -log of that edit-distance:

$$-\log(d(A_1, A_2)) = \bigoplus_{\substack{\log \\ x,y}} [\![A_1]\!](x) + [\![A_2]\!](y) - \log d(x, y)$$

We first show that the cost of the alignment of two strings can be computed using a simple weighted transducer over the log semiring.

6.1 Alignment costs in the log semiring

Let Ψ be the formal power series defined over the alphabet Ω and the log semiring by: $(\Psi, (a, b)) = -\log(c((a, b)))$ for $(a, b) \in \Omega$, and let S be the formal power series S over the log semiring defined by:

$$S = \Omega^* + \Psi + \Omega^*$$

S is a rational power series as a $+$-product and closure of the polynomial power series Ω and Ψ [28, 4]. Thus, by the theorem of Schützenberger [29], there exists a weighted automaton A defined over the alphabet Ω and the semiring $\mathcal{L}$ realizing S. A can also be viewed as a weighted transducer T with input and output alphabets Σ. Figure 5 shows the simple finite-state transducer T realizing S in the particular case of the edit cost function c_0 and with $\Sigma = \{a, b\}$.

Lemma 8. *Let $\omega = (a_0, b_0) \cdots (a_n, b_n) \in \Omega^*$ be an alignment, then (S, ω) is equal to* -log *of the cost of ω.*

Proof. By definition of the $+$-multiplication of power series in the log semiring:

$$\begin{aligned} (S, \omega) &= \bigoplus_{\substack{\log \\ u\,(a_i, b_i)\,v = \omega}} (\Omega^*, u) + (\Psi, (a_i, b_i)) + (\Omega^*, v) \\ &= \bigoplus_{\substack{\log \\ i=0}}^{n} -\log(c((a_i, b_i))) = -\log(\sum_{i=0}^{n} c((a_i, b_i))) = -\log c(\omega) \end{aligned}$$

This proves the lemma. □

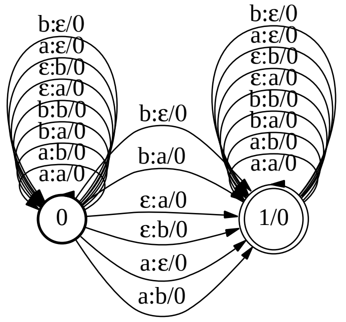

Fig. 5. Weighted transducer over the log semiring representing S with the edit cost function c_0 and $\Sigma = \{a, b\}$. The transition weights and the final weight at state 1 are all equal to 0, since $-\log(c_0(x, y)) = 0$ for $x \neq y$.

The lemma is a special instance of a more general property that can be easily proved in the same way: given an alphabet Σ and a rational set $X \subseteq \Sigma^*$, the power series $\Sigma^* + X + \Sigma^*$ over the log semiring is rational and associates to each string $x \in \Sigma^*$ -log of the number of occurrences of an element of X in x.

6.2 Algorithm

Let A_1 and A_2 be two acyclic weighted automata defined over the alphabet Σ and the log semiring $\mathcal{L}$, and let T be the weighted transducer over the log semiring associated to S.

Let $M = A_1 \circ T \circ A_2$. M can be viewed as a weighted automaton over the alphabet Ω.

Lemma 9. *Let $\omega \in \Omega^*$ be an alignment such that $h(w) = (x, y)$ with $x \in Dom(A_1)$ and $y \in Dom(A_2)$, then:*

$$[\![M]\!](w) = -\log c(\omega) + [\![A_1]\!](x) + [\![A_2]\!](x)$$

Proof. By definition of composition, $[\![M]\!](w)$ represents the value associated by S to the alignment ω with weight $W = [\![A_1]\!](x) + [\![A_2]\!](x)$. By lemma 8 and the definition of power series, S associates to an alignment ω with weight W the following: $-\log c(\omega) + W$. $\square$

The automaton M may contain several paths labeled with the same alignment ω. M is acyclic as the result of the composition of T with acyclic automata, thus it can be determinized. Denote by $\det_{\mathcal{L}}(M)$ the result of that determinization in the log semiring. By definition of determinization, $\det_{\mathcal{L}}(M)$ is equivalent to M but contains exactly one path for each alignment ω between two strings $x \in Dom(A_1)$ and $y \in Dom(A_2)$.

We need to keep for each pair of strings x and y only one path, the one corresponding to the alignment ω of x and y with the minimal cost $c(\omega)$ or equivalently maximal $[\![M]\!](w)$. We will use determinization in the tropical semiring, $\det_{\mathcal{T}}$, to do so. However, to apply this algorithm we first need to ensure that the transducer is normalized so that paths corresponding to different alignments ω but with the same $h(\omega)$ be merged by the automata determinization $\det_{\mathcal{T}}$. By lemma 6, one way to normalize the automaton consists of using the synchronization algorithm, synch, followed by ϵ-removal in the log semiring, $\mathrm{rm}\epsilon_{\mathcal{L}}$.

Theorem 3. *Let N be the deterministic weighted automaton defined by:*

$$-\det_{\mathcal{T}}(-\mathrm{rm}\epsilon_{\mathcal{L}}(\mathrm{synch}(\det_{\mathcal{L}}(A_1 \circ T \circ A_2))))$$

Then for any $x \in Dom(A_1)$ and $y \in Dom(A_2)$:

$$[\![N]\!](x,y) = [\![A_1]\!](x) + [\![A_2]\!](y) - \log d(x,y)$$

Proof. Let $\omega \in \Omega^*$ be an alignment such that $h(w) = (x,y)$ with $x \in Dom(A_1)$ and $y \in Dom(A_2)$, then, by lemma 8:

$$\mathrm{rm}\epsilon_{\mathcal{L}}(\mathrm{synch}(\det_{\mathcal{L}}(A_1 \circ T \circ A_2)))(\omega) = [\![A_1]\!](x) + [\![A_2]\!](y) - \log c(\omega)$$

Since $\mathrm{rm}\epsilon_{\mathcal{L}}(\mathrm{synch}(\det_{\mathcal{L}}(A_1 \circ T \circ A_2)))$ is normalized, by definition of determinization in the tropical semiring, for any $x \in Dom(A_1)$ and $y \in Dom(A_2)$:

$$\begin{aligned}[\![N]\!](x,y) &= \max_{h(\omega)=(x,y)} [\![A_1]\!](x) + [\![A_2]\!](y) - \log c(\omega) \\ &= [\![A_1]\!](x) + [\![A_2]\!](y) - \log d(x,y)\end{aligned}$$

□

Since N is deterministic when viewed as a weighted automaton, the shortest distance from the initial state i to the final states F in the log semiring is exactly what we intended to compute:

$$\bigoplus_{\pi \in P(i,F)} w[\pi] = \bigoplus_{\substack{\log \\ x,y}} [\![A_1]\!](x) + [\![A_2]\!](y) - \log d(x,y) = -\log(d(A_1, A_2))$$

This shortest distance can be computed in linear time using a generalization of the classical single-source shortest paths algorithm for acyclic graphs [20]. Thus, the theorem shows that the edit-distance of two automata A_1 and A_2 can be computed exactly using general weighted automata algorithms.

The worst case complexity of the algorithm is exponential but in practice several techniques can be used to improve its efficiency. First, a heuristic pruning can be used to reduce the size of the original automata A_1 and A_2 or that of intermediate automata and transducers in the algorithm described. Additionally, weighted minimization in the tropical and log semirings [19] can be used to optimally reduce the size of the automata after each determinization. Finally, the automaton A is not determinizable in the log semiring but it can be approximated by a deterministic one for example by limiting the number of insertions, deletions or substitutions to some large but fixed number or by using ϵ-determinization [19]. The advantage of a deterministic A is that it is unambiguous and thus it leads to an unambiguous machine M in the sense that no two paths of M correspond to the same alignment. Thus, it is not necessary to apply determinization in the log semiring, $\det_{\mathcal{L}}$, to M.

6.3 Edit-distance weighted automaton

In some applications such as speech recognition, one might wish to compute not just the edit-distance of A_1 and A_2 but an automaton A_3 accepting exactly the same strings as A_1 and such that the weight associated to $x \in Dom(A_3)$ is -log of the expected edit-distance of x to A_2: $[\![A_3]\!](x) = -\log d(x, A_2)$. In such cases, the automaton A_1 is typically assumed to be unweighted: $[\![A_1]\!](x) = 0$ for all $x \in Dom(A_1)$.

More precisely, A_2 is then the weighted automaton, or word lattice, output of the recognizer, and the weight of each sentence is -log of the probability of that sentence given the acoustic information. However, the word-accuracy of a speech recognizer is measured by computing the edit-distance of the sentence output of the recognizer and the reference sentence [18]. This motivates the algorithm presented in this section. Assuming that all candidate sentences are represented by some automaton A_1 (A_1 could represent all possible sentences for example or just the sentences accepted by A_2), one wishes to determine for each sentence in A_1 its expected edit-distance to A_2 and thus to compute A_3.

Let proj_1 be the operation that creates an acceptor from a weighted transducer by removing its output labels. The following theorem gives the algorithm for computing A_3 based on classical weighted automata algorithms.

Theorem 4. *Let A_1 be an unweighted automaton and A_2 an acyclic weighted automaton over the* log *semiring. Then the edit-distance automaton A_3 can be computed as follows from N:*

$$A_3 = \det_{\mathcal{L}}(\mathrm{proj}_1(N))$$

Proof. Since A_1 is unweighted, by theorem 3, for any $x \in Dom(A_1)$ and $y \in Dom(A_2)$:

$$[\![N]\!](x, y) = [\![A_2]\!](y) - \log d(x, y)$$

To construct A_3 we can omit the output labels of N. $\mathrm{proj}_1(N)$ may have several paths labeled with the same input x. If we apply weighted determinization in the log semiring to it, then by definition the weight of a path labeled with x will be exactly:

$$\begin{aligned}
\bigoplus_{\substack{\log \\ y \in Dom(A_2)}} [\![A_2]\!](y) - \log d(x, y) &= -\log[\sum_{y \in Dom(A_2)} \exp(-[\![A_2]\!](y) + \log d(x, y))] \\
&= -\log[\sum_{y \in Dom(A_2)} \exp(-[\![A_2]\!](y)) d(x, y)] \\
&= -\log d(x, A_2)
\end{aligned}$$

This proves the theorem. □

The weighted automaton A_3 can be further minimized using weighted minimization to reduce its number of states and transitions [19].

In the log semiring, the weight associated to an alignment with cost zero is $\infty = -\log 0$. Thus, paths corresponding to the best alignments would simply not appear in the result. To avoid this effect, one can assign an arbitrary large cost to perfect alignments.

In speech recognition, using a sentence with the lowest expected word error rate instead of one with the highest probability can lead to a significant improvement of the word accuracy of the system [18]. That sentence is simply the label of a shortest path in A_3 and can therefore be obtained from A_3 efficiently using a classical single-source shortest-paths algorithm.

Speech recognition systems often 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 or by using information sources of a different nature. The weighted automaton or word lattice A_3 can be used advantageously for such rescoring purposes.

7 Conclusion

Algorithms for computing the edit-distance of unweighted and weighted automata were given. These algorithms are based on general and efficient weighted automata algorithms over different semirings and classical single-source shortest-paths algorithms. They demonstrate the power of both automata and semiring theories and provide a complex example of the use of multiple semirings in a single application.

The algorithms presented have applications in many areas ranging from text processing to computational biology. They can lead to significant improvements of the word accuracy in large-vocabulary speech recognition as already mentioned (section 6). More recently, the notion of edit-distance was used for the definition of the string kernels used in statistical learning techniques such as support vector machines for classification of biological sequences [13, 33, 8]. These string kernels are specific instances of the more general class of weighted rational relations and can all be computed efficiently using the general algorithm for computing the edit-distance of unweighted automata (section 5). String kernels can be generalized to deal properly with probabilistic or weighted sequences. An algorithm for computing such generalized kernels based on general weighted automata algorithms was given in section 6.

Acknowledgments

I thank Cyril Allauzen and Michael Riley for discussions about this work.

References

1. 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.
2. Marie-Pierre Béal and Olivier Carton. Asynchronous sliding block maps. *Informatique Théorique et Applications*, 34(2):139–156, 2000.
3. Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbucher: Stuttgart, 1979.
4. Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Springer-Verlag: Berlin-New York, 1988.
5. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press: Cambridge, MA, 1992.
6. Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, 1994.
7. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1959.
8. R. Durbin, S.R. Eddy, A. Krogh, and G.J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge UK, 1998.
9. Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.
10. Christiane Frougny and Jacques Sakarovitch. Synchronized Rational Relations of Finite and Infinite Words. *Theoretical Computer Science*, 108(1):45–82, 1993.
11. Vaibhava Goel and William Byrne. Task Dependent Loss Functions in Speech Recognition: A* Search over Recognition Lattices. In *Proceedings of Eurospeech'99, Budapest, Hungary*, 1999.
12. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge, UK., 1997.
13. David Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, 1999.
14. Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, and Setsuo Arikawa. Construction of the CDAWG for a Trie. In *Proceedings of the Prague Stringology Conference (PSC'01)*. Czech Technical University, 2001.

15. Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages.* Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1986.
16. Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston, 1976.
17. Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics - Doklady*, 10:707–710, 1966.
18. Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding Consensus Among Words: Lattice-Based Word Error Minimization. In *Proceedings of Eurospeech'99, Budapest, Hungary*, 1999.
19. Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2, 1997.
20. Mehryar Mohri. General Algebraic Frameworks and Algorithms for Shortest-Distance Problems. Technical Memorandum 981210-10TM, AT&T Labs - Research, 62 pages, 1998.
21. Mehryar Mohri. Generic Epsilon-Removal and Input Epsilon-Normalization Algorithms for Weighted Transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.
22. Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Weighted Automata in Text and Speech Processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language, Budapest, Hungary.* ECAI, 1996.
23. Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32, January 2000.
24. Mehryar Mohri and Richard Sproat. An Efficient Compiler for Weighted Rewrite Rules. In 34*th Meeting of the Association for Computational Linguistics (ACL '96), Proceedings of the Conference, Santa Cruz, California.* ACL, 1996.
25. 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*, AT&T Labs – Research, 1997.
26. Fernando C. N. Pereira and Michael D. Riley. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, Massachusetts, 1997.
27. Eric S. Ristad and Peter N. Yianilos. Learning string edit distance. *IEEE Trans. PAMI*, 20(5):522–532, 1998.
28. Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series.* Springer-Verlag: New York, 1978.
29. Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4, 1961.
30. Marcel Paul Schützenberger. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4(1):47–57, 1977.
31. Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.
32. Robert A. Wagner and Michael J. Fisher. The string to string correction problem. *Journal of the Association for Computing Machinery (ACM)*, 21(1):168–173, 1974.
33. Chris Watkins. Dynamic alignment kernels. Technical Report CSD-TR-98-11, Royal Holloway, University of London, 1999.