Weighted Automata Algorithms

Mehryar Mohri[1,2]

[1] Courant Institute of Mathematical Sciences
251 Mercer Street, New York, NY 10012
`mohri@cims.nyu.edu`

[2] Google Research
76 Ninth Avenue, New York, NY 10011
`mohri@google.com`

This chapter presents several fundamental algorithms for weighted automata and transducers. While the mathematical counterparts of weighted transducers, *rational power series*, have been extensively studied in the past [22, 54, 13, 36], several essential weighted transducer algorithms, e.g., composition, determinization, minimization, have been devised only in the last decade [38, 43], in part motivated by novel applications in speech recognition, speech synthesis, machine translation, other areas of natural language processing, image processing, optical character recognition, and more recently machine learning.

These algorithms can be viewed as the generalization to the weighted transducer case of the standard algorithms for unweighted acceptors. However, this generalization is often not straightforward and has required a number of specific studies either because the old schema could not be applied in the presence of weights and a novel technique was required, as in the case of composition [50, 46], or because of the analysis of the conditions of application of an algorithm as in the case of determinization [38, 3].

The chapter favors a presentation of weighted automata and transducers in terms of graphs, the natural concepts for an algorithmic description and complexity analysis. Also, while power series lead to more concise and rigorous proofs in most cases [36], proofs related to questions of ambiguity naturally require the introduction of paths and reasoning on graph concepts.

1 Preliminaries

This section introduces the definitions and notation related to *weighted finite-state transducers*, weighted transducers for short, and *weighted automata*.

Semiring	Set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+ \cup \{+\infty\}$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\min$	$+$	$+\infty$	0

Table 1. Semiring examples. $\oplus_{\log}$ is defined by: $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

1.1 Semirings

For various operations to be well-defined, the weight set associated to a weighted transducer must have the structure of a *semiring* (see [20]). A system $(S, \oplus, \otimes, \overline{0}, \overline{1})$ is a *semiring* if $(S, \oplus, \overline{0})$ is a commutative monoid with identity element $\overline{0}$, $(S, \otimes, \overline{1})$ is a monoid with identity element $\overline{1}$, $\otimes$ distributes over $\oplus$, and $\overline{0}$ is an annihilator for $\otimes$: for all $a \in S, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. Thus, a semiring is a ring that may lack negation.

Table 1 lists several semirings. In addition to the Boolean semiring, and the probability semiring used to combine probabilities, two semirings often used in applications are the *log semiring*, which is isomorphic to the probability semiring via the negative-log morphism, and the *tropical semiring*, which is derived from the log semiring using the *Viterbi approximation*. In the following definitions, S will be used to denote a semiring.

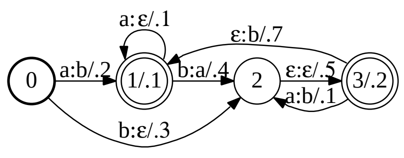
A semiring is said to be *commutative* when the multiplicative operation $\otimes$ is commutative. The semirings listed in Table 1 are all commutative. It is said to be *idempotent* if $x \oplus x = x$ for all $x \in S$. The Boolean semiring and the tropical semiring are idempotent.

1.2 Weighted Transducers and Automata

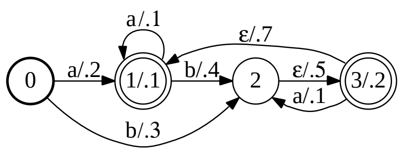
Given an alphabet Σ, we will denote by $|x|$ the length of a string $x \in \Sigma^*$ and by ϵ the *empty string* for which $|\epsilon| = 0$. The *mirror image* of a string $x = x_1 \cdots x_n$ is the string $x^R = x_n x_{n-1} \cdots x_1$.

Finite-state transducers are finite automata in which each transition is augmented with an output label in addition to the familiar input label [12, 22, 54, 36]. Output labels are concatenated along a path to form an output sequence and similarly with input labels. *Weighted transducers* are finite-state transducers in which each transition carries some weight in addition to the input and output labels [54, 36, 52]. The weights are elements of a semiring $(S, \oplus, \otimes, \overline{0}, \overline{1})$.

The $\otimes$-operation is used to compute the weight of a path by $\otimes$-multiplying the weights of the transitions along that path. The $\oplus$-operation computes the weight of a pair of input and output strings (x, y) by $\oplus$-summing the weights of the paths labeled with (x, y). The following gives a formal definition of weighted transducers.



(a)



(b)

Fig. 1. (a) Example of a weighted transducer T over the probability semiring. (b) Example of a weighted automaton A over the probability semiring. A can be obtained from T by removing output labels. A bold circle indicates an initial state with initial weight $\overline{1}$ and a double-circle a final state. A final state q's weight $\rho(q)$ is indicated after the slash symbol representing the state number.

Definition 1. *A* weighted transducer T *over a semiring* $(S, \oplus, \otimes, \overline{0}, \overline{1})$ *is an 8-tuple* $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ *where* Σ *is a finite input alphabet,* Δ *a 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 *a finite multiset*[3] *of transitions, which are elements of* $Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times S \times Q$, $\lambda : I \rightarrow S$ *an initial weight function, and* $\rho : F \rightarrow S$ *a final weight function mapping* F *to* S.

For a state $q \in Q$, we will denote by $E[q]$ the outgoing transitions of q and more generally by $E[Q']$, the outgoing transitions of all states q in a subset of states $Q' \subseteq Q$. An ϵ-transition is a transition with both input and output label equal to ϵ.

A path π of a transducer is an element of E^* with consecutive transitions. We denote by $p[\pi]$ its origin or previous state and by $n[\pi]$ its destination or next state. A cycle π is a path with $p[\pi] = n[\pi]$. An ϵ-cycle is a cycle with both input and output label equal to ϵ. We also denote by

- $P(Q_1, Q_2)$, the set of all paths from a subset $Q_1 \subseteq Q$ to a subset $Q_2 \subseteq Q$.
- $P(Q_1, x, Q_2)$ the subset of all paths of $P(Q_1, Q_2)$ with input label x.
- $P(Q_1, x, y, Q_2)$ the subset of all paths of $P(Q_1, x, Q_2)$ with output label y.

A path in $P(I, F)$ is said to be *accepting* or *successful*. The weight of a path π obtained by $\otimes$-multiplying the weights of its constituent transitions is denoted by $w[\pi]$. For any transducer T, we denote by T^{-1} its *inverse*, that is the transducer obtained from T by swapping the input and output label of each transition.

A transducer T is said to be *regulated* if the output weight associated by T to any pair of strings $(x, y) \in \Sigma^* \times \Delta^*$ defined as:

[3] Thus, there can be two transitions from state p to state q with the same input and output label, and even the same weight. In practice, this is avoided by keeping only one such transition whose weight is the $\oplus$-sum of the weights of the original redundant transitions. We will denote by $\uplus$ the standard join operation of multisets as in $\{1, 2\} \uplus \{1, 3\} = \{1, 1, 2, 3\}$.

$$T(x,y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) \tag{1}$$

is an element of S and its definition does not depend on the order of the terms in the $\oplus$-sum. $T(x,y)$ is defined to be $\overline{0}$ when $P(I,x,y,F) = \emptyset$.[4] Note that in the absence of ϵ-cycles, the set of accepting paths $P(I,x,y,F)$ is finite for any $(x,y) \in \Sigma^* \times \Delta^*$ and thus T is regulated. Also, as we shall see later, in some semirings, such as the four semirings of Table 1, all weighted transducers are regulated. The weighted transducers we will be considering in this chapter will be regulated. Figure 1(a) shows an example of a weighted transducer.

While our definition allows for multiple initial states with initial weights, in all our examples there will be a unique initial state with initial weight $\overline{1}$ and thus that weight is not indicated in figures. Since any weighted transducer can be represented by an equivalent one with this property, this does not represent a real limitation.

A state $q \in Q$ is said to be *non-accessible* (*non-coaccessible*) when there is no path from I to q (resp. from q to F). Non-accessible and non-coaccessible states are called *useless states*. They can be removed using a connection (or trimming) algorithm in linear time without affecting the weight T associates to any pair. A transducer with no useless state is said to be *trim*.

A transducer is said to be *unambiguous* if for any string $x \in \Sigma^*$ it admits at most one accepting path with input label x. It is said to be *deterministic* or *sequential* if it has at most one initial state and at any state no two outgoing transitions share the same input label.

A *weighted automaton* A can be defined as a weighted transducer with identical input and output labels, for any transition. Thus, only string pairs of the form (x,x) can have a non-zero weight by A, which is why the weight associated by A to (x,x) is abusively denoted by $A(x)$ and identified with the *weight associated by A to x*. Similarly, in the graph representation of weighted automata, the output (or input) label is omitted. Figure 1(b) shows an example of a weighted automaton. The language accepted by A is the one accepted by the unweighted automaton obtained by ignoring its weights and is denoted by $L(A)$.

Note that (unweighted) finite automata [51] can be viewed as weighted automata over the Boolean semiring and, similarly, (unweighted) finite-state transducers [22, 54, 12, 36] as weighted transducers defined over the Boolean semiring.

[4] Our definition of *regulated transducers* is more general that the standard one which assumes that transducers do not have cycles with input or output ϵ [54, 36, 52]. The usual definition leads to a simpler presentation but it rules out weighted transducers that are crucial in applications or that can be obtained as a result of application of various algorithms.

2 Shortest-distance algorithms

Shortest-paths problems are familiar problems in computer science and mathematics. In these problems, edge weights may represent distances, costs, or any other real-valued quantity that can be added along a path, and that one may wish to minimize. Thus, edge weights are real numbers and the specific operations used are addition to compute the weight of a path and minimum to select the best path weight.

This section introduces a generalization of this problem to the case where the operations are those of a semiring. These problems turn out to be crucial in the design of several algorithms such as ϵ-removal or pushing and in many other contexts. Different algorithmic solutions will be presented depending on the semiring properties.

We will consider directed graphs $G = (Q, E, w)$ over a semiring S, where Q is a set of vertices, E a set of edges, and $w : E \to S$ the edge weight function which we can extend to any path $\pi = e_1 \ldots e_k$ by $w[\pi] = \bigotimes_{i=1}^{k} w[e_i]$.

2.1 All-Pairs Shortest-Distance Problems

The general all-pairs shortest-distance algorithm described in this section is defined for any *complete semiring*.[5]

Complete Semirings

A semiring $(S, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *complete* if for any index set I and any family $(a_i)_{i\in I}$ of elements of S, $\bigoplus_{i\in I} a_i$ is an element of S whose definition does not depend on the order of the terms in the $\oplus$-sum and that has the following properties [22, 20]:

$$\bigoplus_{i\in I} a_i = \overline{0} \qquad \text{if } \operatorname{card}(I) = 0 \tag{2}$$

$$\bigoplus_{i\in I} a_i = a_i \qquad \text{if } \operatorname{card}(I) = 1 \tag{3}$$

$$\bigoplus_{i\in I} a_i = \bigoplus_{j\in J}\Big(\bigoplus_{i\in I_j} a_i\Big) \qquad \text{for any disjoint partition } I = \bigcup_{j\in J} I_j \tag{4}$$

$$a \otimes \Big(\bigoplus_{i\in I} a_i\Big) = \bigoplus_{i\in I}\big(a \otimes a_i\big) \qquad \text{for any } a \in S \tag{5}$$

$$\Big(\bigoplus_{i\in I} a_i\Big) \otimes a = \bigoplus_{i\in I}\big(a_i \otimes a\big) \qquad \text{for any } a \in S. \tag{6}$$

[5] The algorithm applies in fact more generally to any *closed semiring* as defined in [41], which, unlike the definition given by [17], does not require idempotence. Note that the earlier definition of closed semirings given by Aho et al. [1] is not axiomatically correct (see [37, 26]). Any complete semiring is a closed semiring.

A straightforward consequence of these axioms is that in a complete semiring the identity $\big(\bigoplus_{i \in I} a_i\big)\big(\bigoplus_{j \in J} b_j\big) = \bigoplus_{(i,j) \in I \times J}\big(a_i \otimes b_j\big)$ holds for any two families $(a_i)_{i \in I}$ and $(b_j)_{j \in J}$ of elements of S. Note that in a complete semiring all weighted transducers are regulated since all infinite sums are elements of S.

A complete semiring S is a *starsemiring* [20], that is a semiring that can be augmented with an internal unary *closure* operation * defined by $a^* = \bigoplus_{n=0}^{\infty} a^n$ for any $a \in S$.[6] Furthermore, associativity, commutativity, and distributivity apply to these infinite sums.

The *Boolean semiring* $(\{0,1\}, \vee, \wedge, 0, 1)$ with $a^* = 1$ for $a \in \{0,1\}$, and the *tropical semiring* $(\mathbb{R}_+ \cup \{+\infty\}, \min, +, +\infty, 0)$, with $a^* = 0$ for all $a \in \mathbb{R}_+ \cup \{+\infty\}$, implicitly used in shortest-paths problems, are familiar examples of complete semirings. The more general tropical semiring $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +, +\infty, 0)$ with

$$a^* = \begin{cases} 0 & \text{if } a \in \mathbb{R}_+; \\ -\infty & \text{otherwise,} \end{cases} \tag{7}$$

and $(+\infty) + (-\infty) = (-\infty) + (+\infty) = +\infty$, also defines a complete semiring. Note that the family of complete semirings includes non-idempotent semirings such as the probability semiring $(\mathbb{R}_+ \cup \{+\infty\}, +, \times, 0, 1)$ with the closure operation defined by

$$a^* = \begin{cases} \dfrac{1}{1-a} & \text{if } 0 \leq a < 1; \\ +\infty & \text{otherwise.} \end{cases} \tag{8}$$

The log semiring $(\mathbb{R}_+ \cup \{-\infty, +\infty\}, \oplus_{\log}, +, +\infty, 0)$ which is isomorphic to the probability semiring is also a non-idempotent complete semiring with

$$a^* = \begin{cases} \log(1-a) & \text{if } 0 \leq a < 1; \\ -\infty & \text{otherwise.} \end{cases} \tag{9}$$

The *lattice semiring* $(\mathbb{L}, \vee, \wedge, \perp, \top)$ where $\mathbb{L}$ is a complete and distributive lattice with infimum $\perp$ and supremum $\top$ is a complete semiring with $a^* = \top$ for all $a \in \mathbb{L}$, when it verifies properties (5) and (6) [20]. Thus, all weighted transducers are regulated in the semirings just examined.

All-Pairs Shortest-Distance Algorithm

For a complete semiring, we can define the *distance* or *shortest-distance* from vertex p to vertex q in $G = (Q, E, w)$ by

[6] Thus, with the terminology of [20], it is a *complete starsemiring*. All complete starsemirings are Conway semirings [20].

$$d[p,q] = \bigoplus_{\pi \in P(p,q)} w[\pi], \tag{10}$$

where the $\oplus$-sum runs over the set of all paths from p to q. This definition coincides with the classical definition of shortest-distance where the weights are summed along the path and where the shortest path is sought for the tropical semiring $(\mathbb{R}_+ \cup \{+\infty\}, \min, +, +\infty, 0)$. The general *all-pair shortest-distance problem* is that of computing the shortest distances $d[p,q]$ for all pairs (p,q) with $p, q \in Q$.

This problem can be solved by computing the closure of the matrix $\mathbf{M} = (\mathbf{M}_{pq}) \in S^{|Q| \times |Q|}$ defined by $\mathbf{M}_{pq} = \oplus_{e \in E \cap P(p,q)} w[e]$ for all $p, q \in Q$. Indeed, using the semiring operations in matrix multiplication [20], for $n \in \mathbb{N}$, the coefficient $\mathbf{M}^n_{pq}$ of $\mathbf{M}^n$ gives the $\oplus$-sum of the weights of all paths of length at most n from p to q. For idempotent semirings such as the tropical semiring for which $\overline{1} \oplus x = \overline{1}$ for all $x \in S$, only simple paths (paths with no cycle) need to be considered in the computation of the shortest distances and thus $\mathbf{M}^* = \mathbf{M}^{|Q|-1}$. Using the standard repeated squaring technique [17], $\mathbf{M}^{|Q|-1}$ can be computed in time $\mathbf{\Theta}(|Q|^3(T_\oplus + T_\otimes)\log|Q|)$, where $T_\oplus$ denotes the computational cost of the $\oplus$ operation, and $T_\otimes$ that of the $\otimes$ operation. There exists however a more efficient method for computing all-pairs shortest-distances for all complete semirings based on a generalization of the Floyd-Warshall algorithm.[7]

The Floyd-Warshall algorithm [25, 57] originally designed for the Boolean semiring can be generalized to compute all-pair shortest-distances in all complete semirings. Figure 2 gives the pseudocode of an in-place implementation of the algorithm where $d[i,j]$ corresponds to the tentative shortest distance from vertex i to vertex j. Lines 1-3 initialize each distance $d[i,j]$ to the sum of the weights of the transitions between i and j. By convention, the $\oplus$-sum is $\overline{0}$ if i and j are not adjacent. The loops of lines 4-11 update the tentative shortest-distances in a way that is similar to the steps of the standard Floyd-Warshall algorithm but using operations of an arbitrary complete semiring.

Let T_* denote the cost of the closure operation.

Theorem 2. *Let $G = (Q, E, w)$ be a weighted directed graph over a complete semiring S. Then, the algorithm* Gen-All-Pairs *computes the shortest-distances $d[i,j]$ between all pairs of vertices (i,j) of G in time $\mathbf{\Theta}(|Q|^3(T_\oplus + T_\otimes + T_*))$ and space $\mathbf{\Theta}(|Q|^2)$.*

Proof. Let $P^k(i,j)$ denote the set of paths from i to j with all intermediate vertices within $\{1, \ldots, k\}$. For any $i, j \in Q, k \in \{0\} \cup Q$, let d^k_{ij} be the sum of all paths from i to j with all intermediate vertices within $\{1, \ldots, k\}$: $d^k_{ij} = \bigoplus_{w \in P^k(i,j)} w[\pi]$. Since the semiring is complete, d^k_{ij} is well-defined and in S.

[7] Ésik and Kuich also gave a cubic-time algorithm for computing $\mathbf{M}^*$ for all *Conway semirings* (see [20] for the definition), which include complete semirings [23].

```
Gen-All-Pairs($G$)
 1  for $i \leftarrow 1$ to $|Q|$ do
 2      for $j \leftarrow 1$ to $|Q|$ do
 3          $d[i,j] \leftarrow \bigoplus_{e \in E \cap P(i,j)} w[e]$
 4  for $k \leftarrow 1$ to $|Q|$ do
 5      for $i \leftarrow 1$ to $|Q|, i \neq k$ do
 6          for $j \leftarrow 1$ to $|Q|, j \neq k$ do
 7              $d[i,j] \leftarrow d[i,j] \oplus (d[i,k] \otimes d[k,k]^* \otimes d[k,j])$
 8      for $i \leftarrow 1$ to $|Q|, i \neq k$ do
 9          $d[k,i] \leftarrow d[k,k]^* \otimes d[k,i]$
10          $d[i,k] \leftarrow d[i,k] \otimes d[k,k]^*$
11      $d[k,k] \leftarrow d[k,k]^*$
```

Fig. 2. Generic all-pairs shortest-distance algorithm.

Let π be a path in $P^k(i,j)$. It is either a path from i to j with all intermediate vertices within $\{1,\ldots,k-1\}$ or it can be decomposed into a path from i to k with all intermediate vertices within $\{1,\ldots,k-1\}$, followed by any number of cycles at k with all intermediate vertices in $\{1,\ldots,k-1\}$, followed by a path from k to j with all intermediate vertices within $\{1,\ldots,k-1\}$. Thus, for all $i,j,k \in Q$,

$$P^k(i,j) = P^{k-1}(i,j) \cup (P^{k-1}(i,k)(P^{k-1}(k,k))^* P^{k-1}(k,j)). \tag{11}$$

By definition, a path in $P^{k-1}(i,j)$ does not go through k, thus:

$$P^{k-1}(i,j) \cap (P^{k-1}(i,k)(P^{k-1}(k,k))^* P^{k-1}(k,j)) = \emptyset. \tag{12}$$

Thus, even if S is not idempotent, d^k_{ij} can be decomposed, for all $i,j,k \in Q$, as

$$d^k_{ij} = d^{k-1}_{i,j} \oplus (d^{k-1}_{ik} \otimes (d^{k-1}_{kk})^* \otimes d^{k-1}_{kj}). \tag{13}$$

This identity leads directly to an algorithm for computing all-pairs shortest distances using a triple-indexed array. An in-place implementation of the algorithm limits the space used to that of a single $|Q| \times |Q|$-matrix (Figure 2) and thus the space complexity of the algorithm to $O(|Q|^2)$. The cubic-time complexity follows directly the definition of the algorithm. □

The efficiency of the algorithm can be improved for graphs G with relatively small strongly connected components (SCCs) by decomposing G into its SCCs, which can be done in linear time, then running Gen-All-Pairs on each SCC.

The Gen-All-Pairs algorithm is useful in a variety of applications. With the Boolean semiring, it can be used to compute the transitive closure of any vertex of a graph and then coincides with the classical Floyd-Warshall algorithm [25, 57]. With the tropical semiring, the algorithm can compute the all-pairs shortest distances in the classical case including for graphs with negative cycles using the general topical semiring $(\mathbb{R}\cup\{-\infty,+\infty\},\min,+,+\infty,0)$. The algorithm of [28] based on Dijkstra's algorithm and that of Bellman-Ford has a better time complexity for graphs with real-valued weights, $O(|Q|^2\log|Q|+|Q||E|)$, but it cannot be used with graphs that have a negative cycle. Gen-All-Pairs can also be used to compute the minimum spanning tree of a directed graph using the complete semiring $(\mathbb{R}\cup\{-\infty,\infty\},\min,\max,\infty,-\infty)$ [17]. Finally, it is also useful for computing the epsilon-removal of a weighted automaton in the general case of complete semirings [40] where Johnson's algorithm does not apply, which is the main motivation for our presentation of the algorithm.

Gen-All-Pairs can be used of course to compute single-source shortest distances in graphs G weighted over a complete semiring. The complexity of the Gen-All-Pairs algorithm in this case, $(|Q|^3)$, makes it impractical for large graphs. The next section describes a single-source shortest-distance algorithm which can be significantly more efficient in many cases.

2.2 Single-Source Shortest-Distance Problems

The general single-source shortest-distance algorithm described in this section is defined for any *k-closed semiring* [41, 20].[8]

k-Closed Semirings

Let $k\geq 0$ be an integer. A semiring $(S,\oplus,\otimes,\overline{0},\overline{1})$ is said to be *k-closed* if

$$\forall a\in S,\quad \bigoplus_{n=0}^{k+1} a^n=\bigoplus_{n=0}^{k} a^n. \tag{14}$$

A k-closed semiring is thus a starsemiring with $a^*=\bigoplus_{n=0}^{k} a^n$ for all $a\in S$ (as defined in [20]). The Boolean semiring, the tropical semiring $(\mathbb{R}_+\cup\{+\infty\},\min,+,+\infty,0)$, or $(\mathbb{R}\cup\{-\infty,\infty\},\min,\max,\infty,-\infty)$ are examples of k-closed semirings with $k=0$.

General single-source shortest-distance Algorithm

The shortest-distance $d[i,j]$ from any vertex i to any vertex j is well-defined in a k-closed semiring S. Given a source vertex $s\in Q$, the general single-source shortest-distance problem consists of computing all distances $d[s,q]$, $q\in Q$.

Figure 3 gives the pseudocode of an algorithm computing the single-source shortest-distances for any k-closed semiring [41]. The algorithm is based on a generalization of the relaxation technique to the k-closed semirings.

The algorithm maintains two arrays $d[q]$ and $r[q]$ indexed with vertices. $d[q]$ denotes the tentative shortest distance from the source s to q. $r[q]$ keeps

[8] See also [24, 20] for the related definition of *locally closed semirings*.

```
Gen-Single-Source($G, s$)
 1  for $i \leftarrow 1$ to $|Q|$ do
 2      $d[i] \leftarrow r[i] \leftarrow \overline{0}$
 3  $d[s] \leftarrow r[s] \leftarrow \overline{1}$
 4  $\mathcal{Q} \leftarrow \{s\}$
 5  while $\mathcal{Q} \neq \emptyset$ do
 6      $q \leftarrow$ Head($\mathcal{Q}$)
 7      Dequeue($\mathcal{Q}$)
 8      $r' \leftarrow r[q]$
 9      $r[q] \leftarrow \overline{0}$
10      for each $e \in E[q]$ do
11          if $d[n[e]] \neq d[n[e]] \oplus (r' \otimes w[e])$ then
12              $d[n[e]] \leftarrow d[n[e]] \oplus (r' \otimes w[e])$
13              $r[n[e]] \leftarrow r[n[e]] \oplus (r' \otimes w[e])$
14              if $n[e] \notin \mathcal{Q}$ then
15                  Enqueue($\mathcal{Q}, n[e]$)
```

Fig. 3. Generic single-source shortest-distance algorithm .

track of the sum of the weights $\oplus$-added to $d[q]$ since the last queue extraction of q. The attribute r is needed for the shortest-distance algorithm to work in non-idempotent cases. The algorithm uses a queue $\mathcal{Q}$ to store the set of states to consider for the relaxation steps of lines 11-15 [41]. Any queue discipline, e.g., FIFO, shortest-first, topological (in the acyclic case), can be used.

Different queue disciplines yield different running times for our algorithm. The choice of the best queue discipline to use depends on the semiring and the graph structure.

If the graph is acyclic, then using the topological order queue discipline gives a linear-time algorithm: $O(|Q|+(T_\oplus+T_\otimes)|E|)$. For the tropical semiring $(\mathbb{R}_+ \cup \{+\infty\}, \min, +, +\infty, 0)$ and the best-first queue discipline, the algorithm coincides with Dijkstra's algorithm and its complexity is $O(|E| + |Q| \log |Q|)$ using Fibonacci heaps. In the presence of negative weights but no negative cycles, using a FIFO queue discipline, the algorithm coincides with the Bellman-Ford algorithm.

The initialization step of the algorithm (lines 1-3) takes $O(|Q|)$ time, each relaxation (lines 11-13) takes $O(T_\oplus + T_\otimes + C(A))$ time. There are exactly $N(q)|E[q]|$ relaxations at q. The total cost of the relaxations is thus: $O((T_\oplus + T_\otimes + C(A))|E| \max_{q\in Q} N(q))$. Since each vertex q is inserted in $\mathcal{Q}$ $N(q)$ times (line 15), it is also extracted from $\mathcal{Q}$ $N(q)$ times (lines 6-7), and the general expression of the complexity is

$$O(|Q| + (T_\oplus + T_\otimes + C(A))|E| \max_{q\in Q} N(q) + (C(I) + C(E)) \sum_{q\in Q} N(q)), \quad (15)$$

where $C(E)$ is the worst cost of removing a vertex q from the queue $\mathcal{Q}$, $C(I)$ that of inserting q in $\mathcal{Q}$, and $C(A)$ that of an assignment, including the possible necessary cost of reorganizing the queue.

Theorem 3. *Let $G = (Q, E, w)$ be a weighted directed graph over a k-closed commutative semiring S and let $s \in Q$ be a distinguished source vertex. Then, the algorithm* Gen-Single-Source *computes the single-source shortest-distances $d[s, q]$ to all vertices $q \in Q$ regardless of the queue discipline used for $\mathcal{Q}$.*

The proof of theorem is given in [41].

3 Rational Operations

Regulated weighted transducers are closed under the following three standard operations called *rational operations*:

- the *sum* (or *union*) of two weighted transducers T_1 and T_2 is defined by

$$\forall (x, y) \in \Sigma^* \times \Delta^*,\ (T_1 \oplus T_2)(x, y) = T_1(x, y) \oplus T_2(x, y). \tag{16}$$

- the *product* (or *concatenation*) of two weighted transducers T_1 and T_2 by

$$\forall (x, y) \in \Sigma^* \times \Delta^*,\ (T_1 \otimes T_2)(x, y) = \bigoplus_{\substack{x = x_1 x_2 \\ y = y_1 y_2}} T_1(x_1, y_1) \otimes T_2(x_2, y_2). \tag{17}$$

  The sum runs over all possible ways of decomposing x into a prefix $x_1 \in \Sigma^*$ and a suffix $x_2 \in \Sigma^*$ and similarly $y \in \Delta^*$ into a prefix $y_1 \in \Delta^*$ and a suffix y_2. The product of $n > 0$ instances of T, $\overbrace{T \otimes \cdots \otimes T}^{n}$, is denoted by T^n, and by convention $T^0 = \mathcal{E}$, where $\mathcal{E}$ is the transducer defined by

$$\begin{cases} \mathcal{E}(x, y) = \overline{1} & \text{if } (x, y) = (\epsilon, \epsilon); \\ \overline{0} & \text{otherwise.} \end{cases} \tag{18}$$

- the *closure* (or *Kleene-closure*) of a weighted transducer T is defined by

$$\forall (x, y) \in \Sigma^* \times \Delta^*,\ T^*(x, y) = \bigoplus_{n=0}^{+\infty} T^n(x, y), \tag{19}$$

  when $\bigoplus_{n=0}^{+\infty} T^n(x, y)$ is an element of S for all $(x, y) \in \Sigma^* \times \Delta^*$. Note that in the absence of accepting ϵ-paths, that is when $P(I, \epsilon, \epsilon, F) = \emptyset$, $T^n(x, y) = \overline{0}$ for $n > |x| + |y|$, thus $T^*(x, y)$ is defined by a finite sum and is always an element of S. In complete semirings, the closure operation is defined for all weighted transducers.

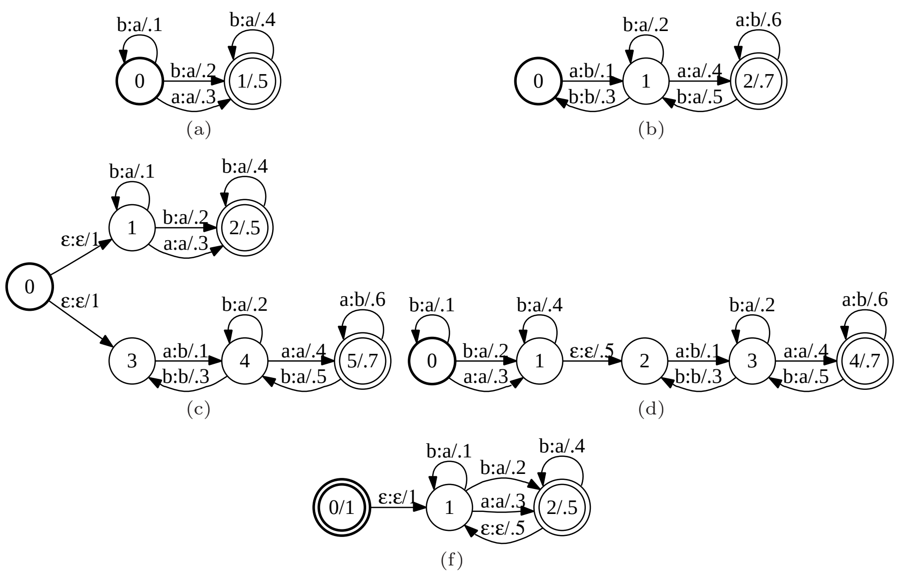


Fig. 4. (a) Weighted transducer T_1 and (b) weighted transducer T_2 over the probability semiring. (c) Sum of T_1 and T_2, $T_1 \oplus T_2$. (d) Product of T_1 and T_2, $T_1 \otimes T_2$. (e) Closure of T_1, T_1^*.

Rational operations can be used to create complex weighted transducers from simpler ones as in the standard case of unweighted acceptors. They admit simple and efficient algorithms. Figures 4(c)-(e) illustrate these algorithms for the particular cases of the transducers T_1 and T_2 of Figures 4(a)-(b).

The transducer sum of two transducers T_1 and T_2 can be constructed from T_1 and T_2 by introducing a new state, made the unique initial state, with ϵ-transitions to the initial states of T_1 and T_2 carrying the weight $\overline{1}$. By construction, the sum of the weights of the paths with input label x and output label y in the resulting transducer is exactly the sum of the weights of the paths with these labels in T_1 and those with these labels in T_2, which matches precisely the definition of $T_1 \oplus T_2$ (Figure 4(c). The time and space complexity of the algorithm is thus linear, $O(|T_1| + |T_2|)$. Furthermore, the algorithm admits a natural *on-demand* or *on-the-fly construction*: states and transitions of the transducer sum can be created only as required by the algorithm using $T_1 \oplus T_2$. This is because the outgoing transition of a state of $T_1 \oplus T_2$ can be constructed only by using that state and T_1 and T_2 without inspecting other states of $T_1 \oplus T_2$. This *local* availability of the information needed to construct the output is what characterizes algorithms admitting natural on-the-fly constructions.

Similarly, the product (or concatenation) of two transducers T_1 and T_2 can be constructed from these transducers by making the final states of T_1 non-final and by creating an ϵ-transition from each final state p of T_1 to each initial state q of T_2 carrying the final weight of p (Figure 4(d)). It is straightforward to verify the correctness of this construction. The time and space complexity of the algorithm is $O(|T_1| + |T_2| + |F_1||I_2|)$. The complexity of the product computation is linear for transducers with a single initial state, which is the typical situation in practice. As with the sum, the product algorithm admits a natural on-demand implementation.

The closure of a transducer T_1 can be constructed as in the standard case of unweighted acceptors. A new initial state is created that is also final with final weight $\overline{1}$. An ϵ-transition with weight $\overline{1}$ is created from this state to the previously initial state of the transducer. Finally, an ϵ-transition is added from each final state p to the previously initial state carrying the final weight of p (Figure 4(e)). The correctness of the construction follows the definition of the closure. The complexity of the algorithm is linear $O(|T_1|)$ and the algorithm admits a natural on-demand implementation as in the case of the other rational operations.

4 Elementary Unary Operations

This section briefly describes three elementary unary operations that are often useful in application.

- the *reversal* of a weighted transducers T produces a transducer T^R that assigns to each pair of strings (x, y) what T assigns to their mirror images (x^R, y^R):
$$T^R(x, y) = T(x^R, y^R). \tag{20}$$
- the *inversion* (or *transposition*) of a weighted transducer T produces a new weighted transducer by swapping the input and output label of each transition
$$T^{-1}(x, y) = T(y, x). \tag{21}$$
- the *projection* of a weighted transducer T on the *input side* (or *left projection*) yields an acceptor $\downarrow T$ by omitting output labels:
$$\downarrow T(x) = \bigoplus_{y} T(x, y). \tag{22}$$
Projection on the output side (or *right projection*), $T\downarrow$, is defined in a similar way.

These operations admit straightforward linear-time algorithms that are illustrated by Figures 5. Inversion and projection are trivial and clearly admit a linear-time algorithm. When the semiring S is commutative, reversal can be obtained by reverting the direction of each transition and making initial states final and final states initial. It can also be obtained as in Figure 5(a)

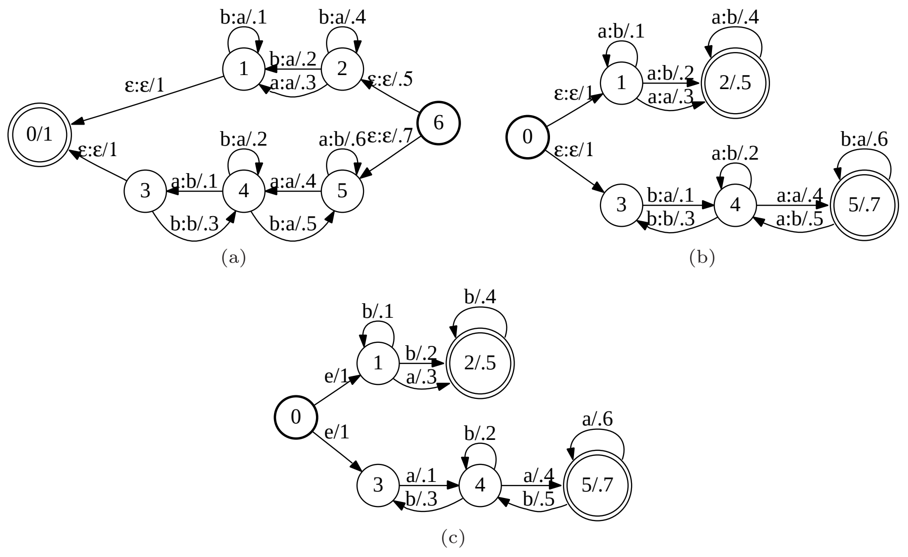


Fig. 5. Elementary operations applies to the transducer $T = T_1 \oplus T_2$ of Figure 4(c). (a) Reversed transducer T^R. (b) Inverted transducer T^{-1}. (c) Projected transducer $\downarrow T$.

by reverting the direction of all transitions, creating a new state p made the unique initial state, with ϵ-transitions to each previously final state q carrying the final weight of q, and making previously initial states final with the same weights. In all cases, reversal does not admit a natural on-demand computation since the computation of the outgoing transitions of a state of the output transducer requires creating or inspecting other output states.

5 Fundamental binary operations

In this section, the semiring S is assumed to be commutative.

5.1 Composition

Composition is a general operation for combining two or more weighted transducers [22, 54, 36, 35]. It is a powerful tool used in a variety of applications to create a complex weighted transducer from simpler ones representing statistical models or discriminative models.

The algorithm for the composition of weighted transducers is a generalization of the standard composition algorithm for unweighted finite-state transducers. However, as we shall see later, the weighted case requires a more subtle technique to deal with ϵ-path multiplicity issues [50, 46]. The algorithm takes as input two weighted transducers

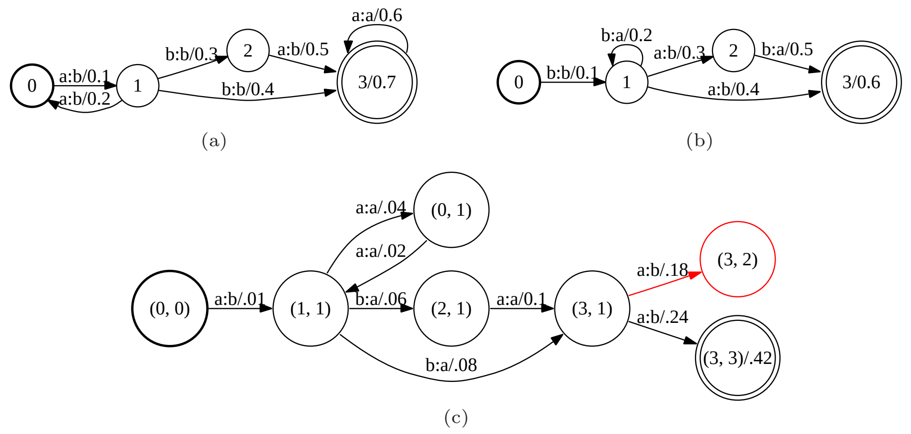


Fig. 6. Weighted transducers (a) T_1 and (b) T_2 over the probability semiring. (c) Illustration of composition of T_1 and T_2, $T_1 \circ T_2$. Some states might be constructed during the execution of the algorithm that are not co-accessible, e.g., $(3, 2)$. Such states and the related transitions can be removed by a trimming (or connection) algorithm in linear-time.

$$T_1 = (\Sigma^*, \Delta^*, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1) \text{ and } T_2 = (\Delta^*, \Omega^*, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2)$$

such that the input alphabet of T_2, Δ, coincides with the output alphabet of T_1, outputs a weighted transducer $T = (\Sigma^*, \Omega^*, Q, I, F, E, \lambda, \rho)$ realizing the composition of T_1 and T_2.

Let T_1 and T_2 be two weighted transducers defined over S such that the input alphabet of T_2 coincides with the output alphabet of T_1. Assume that the infinite sum $\bigoplus_{z \in \Delta^*} T_1(x, z) \otimes T_2(z, y)$ is defined and in S for all $(x, y) \in \Sigma^* \times \Omega^*$. This condition holds for all transducers defined over a complete semiring such as the Boolean semiring, the tropical semiring, the probability semiring and the log semiring, and for all acyclic transducers defined over an arbitrary semiring. Then, the result of the composition of T_1 and T_2 is a weighted transducer denoted by $T_1 \circ T_2$ and defined for all x, y by:

$$(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \Delta^*} T_1(x, z) \otimes T_2(z, y). \tag{23}$$

The sum runs over all strings z labeling a path of T_1 on the output side and a path of T_2 on input label z. The matrix notation we have used emphasizes the connection of composition with matrix multiplication.[9]

There exists a general and efficient algorithm to compute the composition of two weighted transducers. In the absence of ϵs on the input side of T_1 or the output side of T_2, the states of $T_1 \circ T_2$ can be identified with pairs of a

[9] Our choice of a matrix notation as opposed to a functional notation is motivated by its convenience in applications.

state of T_1 and a state of T_2, $Q \subseteq Q_1 \times Q_2$. Initial states are those obtained by pairing initial states of the original transducers, $I = I_1 \times I_2$, and similarly final states are defined by $F = Q \cap (F_1 \times F_2)$. Transitions are obtained by matching a transition of T_1 with one of T_2 from appropriate transitions of T_1 and T_2:

$$E = \biguplus_{\substack{(q_1,a,b,w_1,q_2)\in E_1\\(q_1',b,c,w_2,q_2')\in E_2}} \Big\{\Big((q_1,q_1'),a,c,w_1\otimes w_2,(q_2,q_2')\Big)\Big\}.$$

The following is the pseudocode of the algorithm in the ϵ-free case.

```
WEIGHTED-COMPOSITION($T_1, T_2$)
 1  $Q \leftarrow I_1 \times I_2$
 2  $\mathcal{Q} \leftarrow I_1 \times I_2$
 3  while $\mathcal{Q} \neq \emptyset$ do
 4        $q = (q_1, q_2) \leftarrow$ HEAD($\mathcal{Q}$)
 5        DEQUEUE($\mathcal{Q}$)
 6        if $q \in I_1 \times I_2$ then
 7              $I \leftarrow I \cup \{q\}$
 8              $\lambda(q) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
 9        if $q \in F_1 \times F_2$ then
10              $F \leftarrow F \cup \{q\}$
11              $\rho(q) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$
12        for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$ do
13              if $(q' = (n[e_1], n[e_2]) \notin Q)$ then
14                    $Q \leftarrow Q \cup \{q'\}$
15                    ENQUEUE($\mathcal{Q}, q'$)
16              $E \leftarrow E \uplus \{(q, i[e_1], o[e_2], w[e_1] \otimes w[e_2], q')\}$
17  return $T$
```

E, I, and F are all assumed to be initialized to the empty set. The algorithm uses a queue $\mathcal{Q}$ containing the set of pairs of states yet to be examined. The queue discipline of $\mathcal{Q}$ can be arbitrarily chosen and does not affect the termination of the algorithm. The set of states Q is originally reduced to the set of pairs of the initial states of the original transducers and $\mathcal{Q}$ is initialized to the same (lines 1-2). At each execution of the loop of lines 3-16, a new pair of states (q_1, q_2) is extracted from $\mathcal{Q}$ (lines 4-5). The initial weight of (q_1, q_2) is computed by $\otimes$-multiplying the initial weights of q_1 and q_2 when they are both initial states (lines 6-8). Similar steps are followed for final states (lines 9-11). Then, for each pair of matching transitions (e_1, e_2), a new transition is created according to the rules specified earlier (line 16). If the destination state $(n[e_1], n[e_2])$ has not been found earlier on, it is added to Q and inserted in $\mathcal{Q}$ (lines 14-15).

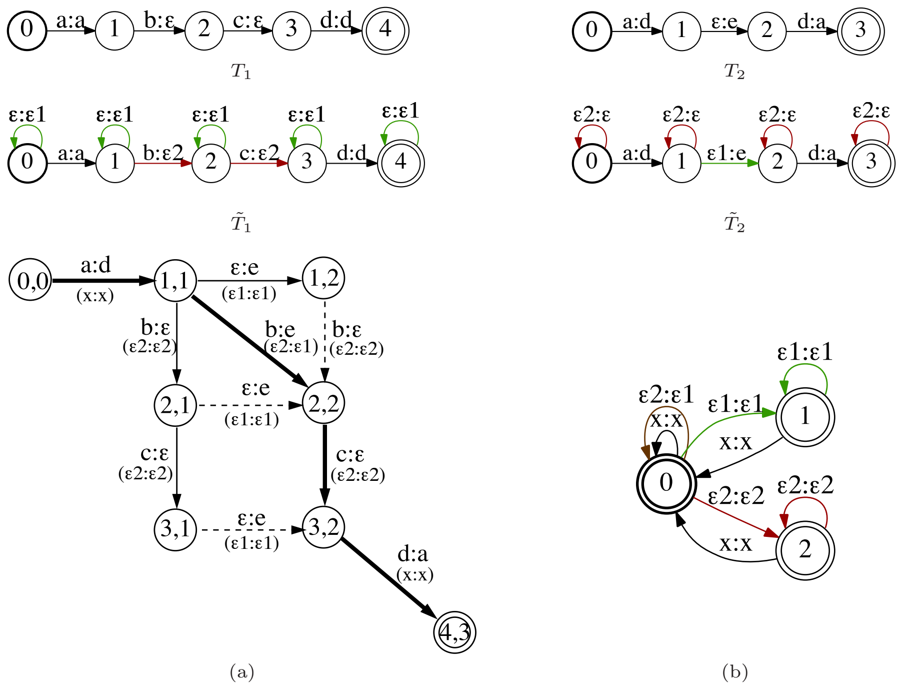


Fig. 7. Redundant ϵ-paths in composition. All transition and final weights are equal to $\overline{1}$. (a) A straightforward generalization of the ϵ-free case would generate all the paths from $(1, 1)$ to $(3, 2)$ when composing T_1 and T_2 and produce an incorrect results in non-idempotent semirings. (b) Filter transducer F [46]. The shorthand x is used to represent an element of Σ.

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: $O(|T_1||T_2|)$. However, an important feature of composition is that it admits a natural on-demand computation which can be used to construct only the part of the composed transducer that is needed. Figures 6(a)-(c) illustrate the algorithm when applied to the transducers of Figures 6(a)-(b) defined over the probability semiring.

More care is needed when T_1 admits output ϵ labels or T_2 input ϵ labels. Indeed, as illustrated by Figure 7, a straightforward generalization of the ϵ-free case would generate redundant ϵ-paths and, in the case of non-idempotent semirings, would lead to an incorrect result. The weight of the matching paths of the original transducers would be counted p times, where p is the number of redundant paths in the result of composition.

To cope with this problem, all but one ϵ-path must be filtered out of the composite transducer. Figure 7 indicates in boldface one possible choice for that path, which in this case is the shortest. Remarkably, that filtering mechanism itself can be encoded as a finite-state transducer F (Figure 7(b)).

To apply that filter, we need to first augment T_1 and T_2 with auxiliary symbols that make the semantics of ϵ explicit. Thus, let $\tilde{T_1}$ ($\tilde{T_2}$) be the weighted transducer obtained from T_1 (resp. T_2) by replacing the output (resp. input) ϵ labels with ϵ_2 (resp. ϵ_1) as illustrated by Figure 7. Thus, matching with the symbol e_1 corresponds to remaining at the same state of T_1 and taking a transition of T_2 with input ϵ. e_2 can be described in a symmetric way. The filter transducer F disallows a matching (ϵ_2, ϵ_2) immediately after (ϵ_1, ϵ_1) since this can be done instead via (ϵ_2, ϵ_1). By symmetry, it also disallows a matching (ϵ_1, ϵ_1) immediately after (ϵ_2, ϵ_2). In the same way, a matching (ϵ_1, ϵ_1) immediately followed by (ϵ_2, ϵ_1) is not permitted by the filter F since a shorter path via the matchings $(\epsilon_2, \epsilon_1)(\epsilon_1, \epsilon_1)$ is possible. Similarly, $(\epsilon_2, \epsilon_2)(\epsilon_2, \epsilon_1)$ is ruled out. It is not hard to verify that the filter transducer F is precisely a finite automaton over pairs accepting the complement of the language

$$L = \sigma^*((\epsilon_1, \epsilon_1)(\epsilon_2, \epsilon_2) + (\epsilon_2, \epsilon_2)(\epsilon_1, \epsilon_1) + (\epsilon_1, \epsilon_1)(\epsilon_2, \epsilon_1) + (\epsilon_2, \epsilon_2)(\epsilon_2, \epsilon_1))\sigma^*,$$

where $\sigma = \{(\epsilon_1, \epsilon_1), (\epsilon_2, \epsilon_2), (\epsilon_2, \epsilon_1), x\}$ [4]. Thus, the filter F guarantees that exactly one ϵ-path is allowed in the composition of each ϵ sequences. To obtain the correct result of composition, it suffices then to use the ϵ-free composition algorithm already described and compute

$$\tilde{T_1} \circ F \circ T_2. \tag{24}$$

Indeed, the two compositions in $\tilde{T_1} \circ F \circ \tilde{T_2}$ no more involve ϵs. Since the size of the filter transducer F is constant, the complexity of general composition is the same as that of ϵ-free composition, that is $O(|T_1||T_2|)$. In practice, the augmented transducers $\tilde{T_1}$ and $\tilde{T_2}$ are not explicitly constructed, instead the presence of the auxiliary symbols is simulated. Further filter optimizations help limit the number of non-coaccessible states created, for example by examining more carefully the case of states with only outgoing non-ϵ-transitions or only outgoing ϵ-transitions [46].

Composition of weighted transducers can be further generalized to the N*-way composition* of weighted transducers [4]. Furthermore, N-way composition of three or more transducers can be substantially faster than the use of the standard composition [5].

5.2 Intersection

The intersection (or Hadamard product) of two weighted automata A_1 and A_2 is defined by [22, 54, 36]:

$$(A_1 \cap A_2)(x) = A_1(x) \otimes A_2(x). \tag{25}$$

It coincides with the special case of composition of weighted transducers where the input label of each transition matches its output label. Thus, the same

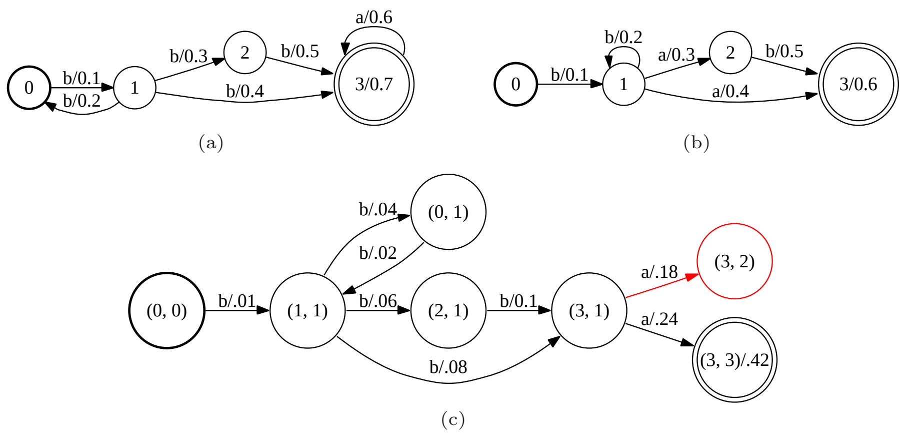


Fig. 8. Weighted Automata (a) A_1 and (b) A_2 over the probability semiring. (c) Illustration of intersection of A_1 and A_2, $A_1 \cap A_2$. Some states might be constructed during the execution of the algorithm that are not co-accessible, e.g., $(3, 2)$. Such states and the related transitions can be removed by a trimming (or connection) algorithm in linear-time.

algorithm can be used to compute intersection with the same complexity. Figure 8 illustrates the application of the algorithm to two weighted automata extracted from the weighted transducers of Figure 6.

5.3 Difference

Negation is not defined for all semirings, but a *difference* operation can be defined for a weighted automata A_1 and an unweighted deterministic automaton A_2 as follows:[10]

$$\forall x \in \Sigma^*, (A_1 - A_2)(x) = \begin{cases} A_1(x) & \text{if } x \notin L(A_2) \\ \overline{0} & \text{otherwise.} \end{cases} \tag{26}$$

Thus, $(A_1 - A_2)$ is the weighted automaton A_1 from which all accepting paths labeled with a string accepted by A_2 are removed, which leads to the following equivalent formulation:

$$\forall x \in \Sigma^*, (A_1 - A_2)(x) = (A_1 \cap \overline{A}_2)(x), \tag{27}$$

where $\overline{A}_2$ is a weighted automaton over the semiring S accepting exactly the complement of $L(A_2)$ and assigning weight $\overline{1}$ to each string accepted. Since A_2 is deterministic, its complement $\overline{A}_2$ can be computed from A_2 in linear time, with the following two steps:

[10] Of course, when negation is defined, $A_1 \oplus (\ominus A_2)$ defined by $\forall x \in \Sigma^*, A_1 \oplus (\ominus A_2) = A_1(x) \ominus A_2(x)$ can be computed by applying the sum algorithm to A_1 and $\ominus A_2$. The semantics of the difference operation considered here is different.

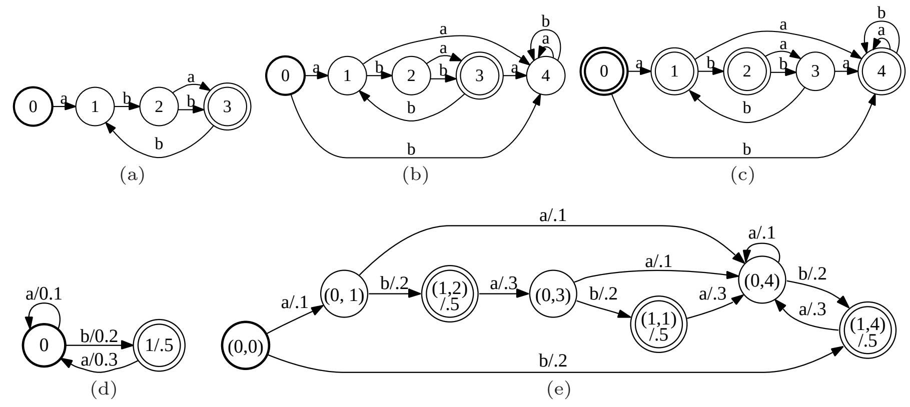


Fig. 9. (a) Unweighted automaton A_2. (b) Complete automaton equivalent to A_2. (c) Complement of A_2, $\overline{A}_2$, all weights are equal to $\overline{1}$ and thus not indicated. (d) Weighted automaton A_1 defined over the probability semiring. (e) Difference of A_1 and A_2, $A_1 - A_2$, obtained by intersection of A_1 and $\overline{A}_2$.

- *completion*: first making A_2 *complete*, that is creating an equivalent automaton to A_2 such that all alphabet symbols can be read from any state. This can be done by augmenting A_2 with a new state p with self-loops labeled with all alphabet symbols, and by adding a transition labeled with $a \in \Sigma$ from state q to p when no transition labeled with a is available at q in A_2.
- *complementation*: then making all final states of the modified automaton A_2 non-final and vice-versa. Finally, all weights of the automaton are set to $\overline{1}$ to make it an automaton over S.

Both of these steps can be executed in linear time $O(|A_2| + |\Sigma|)$ and admit a natural on-demand implementation. Note that the complementation of arbitrary finite automata is PSPACE-complete [1], this is the reason why A_2 was assumed to be deterministic here. The difference can then be obtained by computing the intersection of A_1 and $\overline{A}_2$. Since intersection or composition also admit a natural on-the-demand computation, the same is true of the difference algorithm. Note that using that property, the alphabet symbol actually used in complementation can be limited to the symbols appearing in A_1 and A_2. Thus, the overall complexity of difference is $O(|A_1|(|A_2| + |\Sigma'|)$.

Figure 9 illustrates the difference algorithm.

6 Optimization algorithms

6.1 Epsilon-Removal

The use of various automata or transducer operations such as rational operations generate ϵ-transitions. These transitions cause some delay in the use of the resulting transducers since the search for an alphabet symbol to match in composition or other similar operations requires reading some sequences of ϵs first. To make these weighted transducers more efficient to use, it may be preferable to *remove* all ϵ-transitions, that is to create an equivalent weighted transducer with no ϵ-transition. This section describes a general ϵ-removal algorithm that precisely achieves this task.

Simply removing ϵ-transitions from the input transducer clearly does not result in an equivalent one. Instead, for a given state p, the non-ϵ-transitions of all states q reachable from p via ϵ-transitions should be added to those of p. In the weighted case, this does not result in an equivalent transducer since the weights of the ϵ-transitions from p to q would be ignored. Thus, before adding an outgoing transition of state q to p, the weight it carries must be pre-$\otimes$-multiplied by the sum of the weights of the ϵ-paths from p to q. To ensure that this weight is an element of the semiring, we will assume that S is complete.[11]

This leads to a two-step algorithm [40]. Given a transducer T, let T_ϵ denote the transducer derived from T by keeping only ϵ-transitions and let $d_\epsilon[p, q] = \bigoplus_{\pi \in P(p,\epsilon,q)} w[\pi]$ denote the distance from state p to state q in T_ϵ. Then, the following are the two main steps of the algorithms:

- ϵ-closure computation: at each state p, the *weighted ϵ-closure* defined by
$$C(p) = \{(q, w) : P(p, \epsilon, q) \neq \emptyset, w = d_\epsilon[p, q]\} \tag{28}$$
is computed.
- actual removal of ϵs: all ϵ-transitions are removed and for each p and each $(q, w) \in C(p)$, the transition set of p is augmented with the following transitions
$$\{(p, a, b, d_\epsilon[p, q] \otimes w, r) : (q, a, b, w, r) \in E, (a, b) \neq (\epsilon, \epsilon)\}. \tag{29}$$
If there exists $(q, w) \in C(p)$ with $q \in F$, then $d_\epsilon[p, q] \otimes \rho(q)$ must be $\oplus$-added to the final weight of p.

The following is the pseudocode of the algorithm which follows the main steps just discussed.

Theorem 4 ([40]). *Let T be a weighted transducer over a complete semiring S. Assume that the closures $C(p)$ can be computed for any state p of T. Then, the weighted transducer T' returned by the epsilon-removal algorithm just described is equivalent to T.*

[11] The results presented also hold in the case of closed semirings.

```
EPSILON-REMOVAL($T$)
 1  for each $p \in Q$ do
 2      COMPUTE-CLOSURE($C(p)$)
 3  $E' \leftarrow \overline{E}_\epsilon \leftarrow \{(p, a, b, w, q) \in E : (a, b) \neq (\epsilon, \epsilon)\}$
 4  $F' \leftarrow F$
 5  $\rho' \leftarrow \rho$
 6  for each $p \in Q$ do
 7      for each $(q, w') \in C[p]$ do
 8          $E'[p] \leftarrow E'[p] \uplus \{(p, a, b, w' \otimes w, r) : (q, a, b, w, r) \in \overline{E}_\epsilon\}$
 9          if $q \in F$ then
10              if $p \notin F$ then
11                  $F' \leftarrow F \cup \{p\}$
12                  $\rho'[p] \leftarrow \overline{0}$
13              $\rho'[p] \leftarrow \rho'[p] \oplus (w' \otimes \rho(q))$
14  return $T' = (\Sigma, \Delta, Q, I, F', E', \lambda, \rho')$
```

The proof is simple and is given in [40].

The ϵ-closures $C(p)$ can be computed using an all-pair shortest-distance algorithm over T_ϵ when the semiring S is complete, or by applying the single-shortest distance algorithm from each source p when the semiring is k-closed, as described in Section 2. The complexity of the second stage of the algorithm (lines 6-13) is in $O(|Q|^2 + |Q||E|)$ since in the worst case each $C(p)$ contains all states of the transducer. Thus, the overall complexity of the algorithm is

$$O(|\text{COMPUTE-CLOSURE}| + |Q|^2 + |Q||E|(T_\oplus + T_\otimes)), \tag{30}$$

where $O(|\text{COMPUTE-CLOSURE}|)$ denotes the total cost of the closure computation. We are now examining several special cases of practical interest:

- T_ϵ is acyclic, that is T admits no ϵ-cycle, a rather frequent case in practice. In that case, the single-source shortest distance algorithm can be used for any semiring S and has linear time complexity. The total complexity of applying the algorithm at each state is $O(|Q|^2 + |Q||E|(T_\oplus + T_\otimes))$ and matches that of the second stage. Thus, the overall complexity of epsilon-removal is then
$$O(|Q|^2 + |Q||E|(T_\oplus + T_\otimes)). \tag{31}$$
The algorithm can in fact be improved in that special case. The complexity of the computation of the all-pairs shortest distances can be substantially improved if the states of T_ϵ are visited in reverse topological order and if the single-source shortest-distance algorithm is interleaved with the actual removal of ϵs as follows: for each state p of T_ϵ visited in reverse topological order,
  - run a single-source shortest-distance algorithm with source p to compute the distance from p to each state q in T_ϵ;

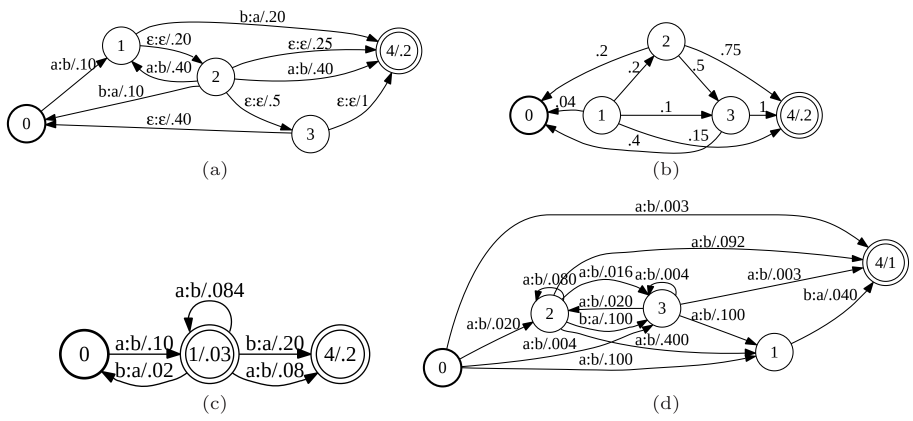


Fig. 10. (a) Weighted transducer T defined over the probability semiring. (b) Weighted graph showing non-$\overline{0}$ all-pair distance $d_\epsilon[p,q]$. From each state p, the outgoing weighted edges give the closure $C(p)$. (c) Weighted transducer T' resulting from T by epsilon-removal. (d) Weighted transducer T'' resulting from T by reverse epsilon-removal.

– then remove ϵ-transitions leaving q and update the final weight as already described.

The reverse topological order guarantees that the ϵ-paths leaving p are reduced to the ϵ-transitions leaving p. Thus, the cost of the shortest-distance algorithm run from p only depends on the number of ϵ-transitions leaving p and the total cost of the computation of the shortest-distances is linear: $O(|Q| + (T_\oplus + T_\otimes)|E|)$.

- S is the tropical semiring. In that case, the complexity of the first stage of the algorithm is that of a standard shortest-path algorithm from each state of T_ϵ. Using Fibonacci heaps, the complexity of the first stage of the algorithm is thus $O(|Q||E| + |Q|^2 \log |Q|)$. Thus, the overall complexity of epsilon-removal is again
$$O(|Q|^2 + |Q||E|(T_\oplus + T_\otimes)). \tag{32}$$
- S is a complete semiring. In that case, when the all-pairs shortest-distance algorithm of Section 2 is the only algorithm available, the complexity of the first stage of the algorithm is $\mathbf{\Theta}(|Q|^3(T_\oplus + T_\otimes + T_*))$ and the overall complexity of epsilon-removal is also
$$O(|Q|^3(T_\oplus + T_\otimes + T_*) + |Q||E|(T_\oplus + T_\otimes)). \tag{33}$$

Epsilon-removal does not create any new state. However, not all states of the original transducer may be necessary. States with only incoming (or outgoing) ϵ-transitions become non-accessible (resp. non-coaccessible) after removal of these transitions, which causes other states not to be accessible or co-accessible. All of these states and corresponding transitions can be removed in linear time using a standard trimming or connection algorithm.

During the epsilon-removal construction, it may happen quite often in practice that several transitions from the same state p to the same state q, with the same input and output label, need to be constructed in the resulting transducer T'. To avoid this redundancy, the weights of these transitions are $\oplus$-summed to maintain at any time a single transition instead.

Note that epsilon-removal admits a natural on-demand computation since the outgoing transitions of state q of the output automaton can be computed directly using the ϵ-closure of q. However, the reverse topological order described in the case of an acyclic T_ϵ requires examining all states of T_ϵ and thus that version of the algorithm cannot be viewed as a natural on-demand construction. In practice, both versions of the algorithm can be useful.

When epsilon-removal is to be followed immediately by the application of determinization, the integration of these two operations often results in much more efficient overall computation. This is because the ϵ-closure of a subset of states created by determinization can be computed by a single shortest-distance algorithm with all states of the subset serving as sources, rather than a distinct one for each state of the subset. Since some states can be reached by several elements of the subset, the first method provides more sharing.

This integration of determinization and epsilon-removal can be extended to the weighted case where the weighted determinization [38] presented in the next section is used.[12]

Epsilon-removal can be straightforwardly modified to remove transitions with input label a and output label b, with $(a, b) \neq (\epsilon, \epsilon)$. This can be done for example by relabeling ϵ-transitions with a new label and replacing (a, b) by (ϵ, ϵ), applying epsilon-removal, and then restoring original ϵs. The resulting transducer is equivalent to the original if (a, b) is assigned the semantics of (ϵ, ϵ).

Figure 10 illustrates the epsilon-removal algorithm. Figure 10(c) shows the transducer T' resulting from the transducer T of Figure 10(a) by application of epsilon-removal. Note that only three of the original states remain in T'. As already discussed, since state 2 and 3 admit only incoming ϵ-transitions (and only outgoing ϵ-transitions in the case of state 3), after removal of ϵ-transitions they become inaccessible and can thus be removed. Figure 10(b) indicates all non-$\overline{0}$ shortest-distances between states in T_ϵ, which summarizes the closure information. These distances are used to determine the weight of the new transitions added.

Instead of removing an ϵ-transition from p to q by adding to state p all non-ϵ-transitions leaving q, one can equivalently proceed by adding to q all non-ϵ-transitions entering p. This is equivalent to applying epsilon-removal in

[12] It is however limited to the cases where the result of epsilon-removal is *determinizable*, that is cases where the determinization algorithm terminates, which, as we shall see later, does not always hold in the weighted case.

the same way as before but to the reverse of T. We will thus refer to *reverse epsilon-removal* as the algorithm that consists of the following sequence of operations: reversal, epsilon-removal, reversal. Figure 10(d) shows T'' the result of the application of reverse epsilon-removal to T. T'' is equivalent to T and T'' and has the same number of states as T, but in this case has more transitions than T'. Which algorithm epsilon-removal or reverse epsilon-removal, produces the smallest transducer depends on the number of outgoing transitions of the states q reached by an ϵ-path in the epsilon-removal case, or the number of incoming transitions of the states p with incoming ϵs in the reverse epsilon-removal case. The decision of the direction of epsilon-removal can be made in fact for each pair of states (p, q) based upon these quantities.

6.2 Determinization

This section describes a general determinization algorithm for weighted automata and transducers [38] which generalizes the standard powerset construction for unweighted finite automata. The presentation will focus on the case of weighted automata, the weighted transducer case can be treated in a similar way or as a special case of the general algorithm we present [38].

A weighted automaton is said to be *deterministic* or *subsequential* 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 called *determinization.* Weighted determinization requires some technical conditions on the semiring or the weighted automaton which we will first introduce. These conditions hold in most cases in practice.

Weighted determinization is a generic algorithm: it works with any *weakly divisible semiring*. A semiring is said to be *divisible* if all non-$\overline{0}$ elements admit an inverse, that is if $S - \{\overline{0}\}$ is a group. $(S, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *weakly divisible* if for any x and y in S such that $x \oplus y \neq \overline{0}$, there exists at least one z such that $x = (x \oplus y) \otimes z$. The $\otimes$-operation is *cancellative* if z is unique and we can write: $z = (x \oplus y)^{-1}x$. When z is not unique, we can still assume that we have an algorithm to find one of the possible z and call it $(x \oplus y)^{-1}x$. Furthermore, we will assume 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 $x, y, u \in S$ such that $u \neq \overline{0}$. A semiring is *zero-sum-free* if for any x and y in S, $x \oplus y = \overline{0}$ implies $x = y = \overline{0}$.

Additionally, we assume that for any string $x \in \Sigma^*$, the sum of the weights of the paths labeled with x and starting at an initial state is non-$\overline{0}$: $w[P(I, x, Q)] \neq \overline{0}$. This condition is always satisfied with trim weighted automata over the tropical semiring or any zero-sum-free semiring.

The pseudocode of the algorithm is given below with Q', I', F', and E' all initialized to the empty set.

A *weighted subset* p' of Q is a set of pairs $(q, x) \in Q \times S$. We will denote by $Q[p']$ the set of states q of the weighted subset p'. $E[Q[p']]$ represents the

```
Weighted-Determinization($A$)
 1  $i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
 2  $\lambda'(i') \leftarrow \overline{1}$
 3  $\mathcal{Q} \leftarrow \{i'\}$
 4  while $\mathcal{Q} \neq \emptyset$ do
 5      $p' \leftarrow$ Head($\mathcal{Q}$)
 6      Dequeue($\mathcal{Q}$)
 7      for each $x \in i[E[Q[p']]]$ do
 8          $w' \leftarrow \bigoplus\{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
 9          $q' \leftarrow \{(q, \bigoplus\{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) :$
                    $q = n[e], i[e] = x, e \in E[Q[p']]\}$
10          $E' \leftarrow E' \cup \{(p', x, w', q')\}$
11          if $q' \notin Q'$ then
12              $Q' \leftarrow Q' \cup \{q'\}$
13              if $Q[q'] \cap F \neq \emptyset$ then
14                  $F' \leftarrow F' \cup \{q'\}$
15                  $\rho'(q') \leftarrow \bigoplus\{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
16              Enqueue($\mathcal{Q}, q'$)
17  return $T'$
```

set of transitions leaving these states, and $i[E[Q[p']]]$ the set of input labels of these transitions.

The states of the output automaton can be identified with (weighted) subsets of the states of the original automaton. A state r of the output automaton that can be reached from the start state by a path π is identified with the set of pairs $(q, x) \in Q \times S$ such that q can be reached from an initial state of the original machine by a path σ with $i[\sigma] = i[\pi]$ and $\lambda(p[\sigma]) \otimes w[\sigma] = \lambda(p[\pi]) \otimes w[\pi] \otimes x$. Thus, x can be viewed as the *residual* weight at state q.

Determinization does not terminate for all weighted automata. As we shall see, not all weighted automata are *determinizable* by the algorithm just described. When it terminates, the algorithm returns a subsequential weighted automaton $A' = (\Sigma, Q', I', F', E', \lambda', \rho')$, equivalent to the input $A = (\Sigma, Q, I, F, E, \lambda, \rho)$.

The algorithm uses a queue $\mathcal{Q}$ containing the set of states of the resulting automaton A', yet to be examined. The queue discipline of $\mathcal{Q}$ can be arbitrarily chosen and does not affect the termination of the algorithm. A' admits a unique initial state, i', defined as the set of initial states of A augmented with their respective initial weights. Its input weight is $\overline{1}$ (lines 1-2). $\mathcal{Q}$ originally contains only the subset i' (line 3). At each execution of the loop of lines 4-16, a new subset p' is extracted from $\mathcal{Q}$ (lines 5-6). For each x labeling at least one of the transitions leaving a state p of the subset p', a new transition with input label x is constructed. The weight w' associated to that transition is the sum of the weights of all transitions in $E[Q[p']]$ labeled with x pre-$\otimes$-multiplied

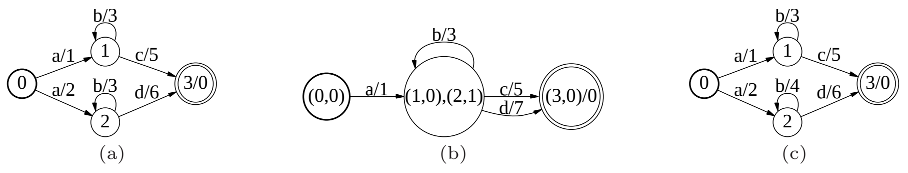


Fig. 11. Determinization of weighted automata. (a) Weighted automaton over the tropical semiring A. (b) Equivalent weighted automaton B obtained by determinization of A. (c) Non-determinizable weighted automaton over the tropical semiring, states 1 and 2 are non-twin siblings.

by the residual weight v at each state p (line 8). The destination state of the transition is the subset containing all the states q reached by transitions in $E[Q[p']]$ labeled with x. The weight of each state q of the subset is obtained by taking the $\oplus$-sum of the residual weights of the states p $\otimes$-times the weight of the transition from p leading to q and by *dividing* that by w'. The new subset q' is inserted in the queue $\mathcal{Q}$ when it is a new state (line 15). If any of the states in the subset q' is final, q' is made a final state and its final weight is obtained by summing the final weights of all the final states in q', pre-$\otimes$-multiplied by their residual weight v (line 14).

Figure 11 illustrates the determinization of a weighted automaton over the tropical semiring. The worst case complexity of determinization is exponential even in the unweighted case. However, in many practical cases such as for weighted automata used in large-vocabulary speech recognition, this blow-up does not occur. It is also important to notice that just like composition, determinization admits a natural lazy implementation which can be useful for saving space.

Unlike the unweighted case, determinization does not halt on all input weighted automata. In fact, some weighted automata, non *subsequentiable* automata, do not even admit equivalent subsequential machines. But even for some subsequentiable automata, the algorithm does not halt. 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.

There exists a general *twins property* for weighted automata that provides a characterization of determinizable weighted automata under some general conditions. Let A be a weighted automaton over a weakly divisible semiring S. Two states q and q' of A are said to be *siblings* if there exist two strings x and y in A^* such that both q and q' can be reached from I by paths labeled with x and there is a cycle at q and a cycle at q' both labeled with y. When S is a commutative and cancellative semiring, two sibling states are said to be *twins* iff for any string y:

$$w[P(q, y, q)] = w[P(q', y, q')] \tag{34}$$

A has *the twins property* if any two sibling states of A are twins.[13] Figure 11(c) shows an unambiguous weighted automaton over the tropical semiring that does not have the twins property: states 1 and 2 can be reached by paths labeled with a from the initial state and admit cycles with the same label b, but the weights of these cycles (3 and 4) are different.

The following theorem is proven in [38].

Theorem 5 ([38]). *Let A be a weighted automaton over the tropical semiring. If A has the twins property, then A is determinizable.*

With trim unambiguous weighted automata, the condition is also necessary [38, 3].

Theorem 6 ([38, 3]). *Let A be a trim unambiguous weighted automaton over the tropical semiring. Then the three following properties are equivalent:*

1. *A is determinizable.*
2. *A has the twins property.*
3. *A is subsequentiable.*

There exists an efficient algorithm for testing the twins property for trim unambiguous and even *cycle-unambiguousweighted automata!cycle-unambiguous* weighted automata in time $O(|Q|^2+|E|^2)$ [3].[14] Note that any acyclic weighted automaton over a zero-sum-free semiring has the twins property and is determinizable.

The existence of an equivalent sequential weighted automaton for a finitely ambiguous weighted automaton over the tropical semiring was shown to be decidable [32]. The twins property has also been shown more recently to be a necessary and sufficient condition for the determinizability of finitely ambiguous trim weighted automata, that is trim automata for which at most a fixed finite number of accepting paths are labeled by any string, that are defined over the tropical semiring of integers $(\mathbb{Z} \cup \{+\infty\}, \min, +, +\infty, 0)$ [31]. A more general notion of *clones property* was introduced by the same author and shown to be a decidable necessary and sufficient condition characterizing determinizability for *polynomially ambiguous* automata over the tropical semiring, that is weighted automata over the tropical semiring for which the number of accepting paths of any string x is bounded by a fixed polynomial defined over the length of x.

[13] The notion of twins property was originally introduced for unweighted finite-state transducers by [15, 16] and was shown to be decidable. Polynomial-time algorithms were later given to test this property for functional transducers in time $O(|Q|^4(|Q|^2+|E|^2)|\Delta|)$ by [58], $O(|Q|^4(|Q|^2+|E|^2))$ by [11], and $O(|Q|^2(|Q|^2+|E|^2))$ by [3], where Q is the set of states of the input transducer, E the set of its transitions and Δ the output alphabet.

[14] An automaton is cycle-unambiguous if for any state q and any string x there exists at most one cycle at q labeled with x.

6.3 Weight pushing

The choice of the distribution of the total weight along each successful path of a weighted automaton does not affect the definition of the function realized by that automaton, but it may have a critical impact on efficiency in many applications, e.g., information extraction or natural language processing, where a heuristic pruning can often be used to visit only a subpart of the automaton. There exists an algorithm, *weight pushing*, for normalizing the distribution of the weights along the paths of a weighted automaton or more generally a weighted directed graph [38, 43].

Let A be a weighted automaton over a semiring S. Assume that S is zero-sum-free and weakly divisible. For any state $q \in Q$, assume that the following sum is defined and in S:

$$d[q] = \bigoplus_{\pi \in P(q,F)} (w[\pi] \otimes \rho(n[\pi])). \tag{35}$$

$d[q]$ is the *shortest-distance* from q to F including the final weight. $d[q]$ is well-defined for all $q \in Q$ when S is a k-closed semiring. The weight pushing algorithm consists of computing each shortest-distance $d[q]$ and of *reweighting* the transition weights, initial weights and final weights in the following way:

$$\forall e \in E \text{ s.t. } d[p[e]] \neq \overline{0}, \quad w[e] \leftarrow d[p[e]]^{-1} \otimes w[e] \otimes d[n[e]] \tag{36}$$

$$\forall q \in I, \quad \lambda(q) \leftarrow \lambda(q) \otimes d[q] \tag{37}$$

$$\forall q \in F \text{ s.t. } d[q] \neq \overline{0}, \quad \rho(q) \leftarrow d[q]^{-1} \otimes \rho(q). \tag{38}$$

Roughly speaking, the algorithm *pushes the weights* of each path as much as possible towards the initial states. Figures 12(a)-(c) illustrate the application of the algorithm in a special case both for the tropical and probability semirings.

Each of the operations described can be assumed to be done in constant time, thus reweighting can be done in linear time $O(T_\otimes |A|)$ where $T_\otimes$ denotes the worst cost of an $\otimes$-operation. The complexity of the computation of the shortest-distances depends on the semiring and the algorithm used (see Section 2). In the case of k-closed semirings such as the tropical semiring, $d[q]$ can be computed using a single-source shortest-path algorithm. The complexity of the algorithm is linear in the case of an acyclic automaton: $O(|Q| + (T_\oplus + T_\otimes)|E|)$, where $T_\oplus$ denotes the worst cost of an $\oplus$-operation. In the case of a general weighted automaton over the tropical semiring, the complexity of the algorithm is $O(|E| + |Q| \log |Q|)$.

In the case of complete semirings such as $(\mathbb{R}_+, +, \times, 0, 1)$, a generalization of the Floyd-Warshall algorithm for computing all-pairs shortest-distances can be used. The complexity of the algorithm is $\mathbf{\Theta}(|Q|^3(T_\oplus + T_\otimes + T_*))$ where T_* denotes the worst cost of the closure operation. The space complexity of these algorithms is $\mathbf{\Theta}(|Q|^2)$. These complexities make it impractical to use the Floyd-Warshall algorithm for computing $d[q]$, $q \in Q$, for relatively

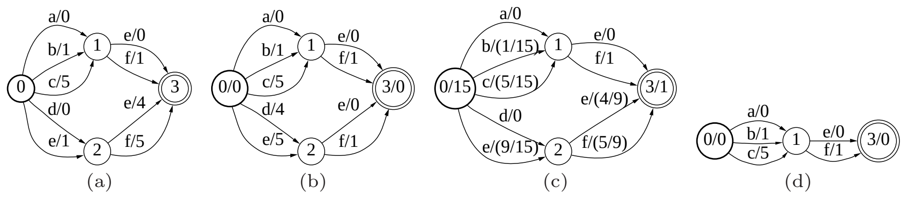


Fig. 12. Weight pushing algorithm. (a) Weighted automaton A. (b) Equivalent weighted automaton B obtained by weight pushing in the tropical semiring. (c) Weighted automaton C obtained from A by weight pushing in the probability semiring. (d) Minimal weighted automaton over the tropical semiring equivalent to A.

large graphs or automata of several hundred million states or transitions. An approximate version of a generic shortest-distance algorithm can be used instead to compute $d[q]$ efficiently.

Note that if $d[q] = \overline{0}$, then, since S is zero-sum-free, the weight of all paths from q to F is $\overline{0}$. Let A be a weighted automaton over the semiring S. Assume that S is complete or k-closed and that the shortest-distances $d[q]$ are all well-defined and in $S - \{\overline{0}\}$. Note that in both cases we can use the distributivity over the infinite sums defining shortest distances. Let e' (π') denote the transition e (path π) after application of the weight pushing algorithm. e' (π') differs from e (resp. π) only by its weight. Let λ' denote the new initial weight function, and ρ' the new final weight function.

The following proposition is proven in [38, 43].

Proposition 7 ([38, 43]). *Let $B = (A, Q, I, F, E', \lambda', \rho')$ be the result of the weight pushing algorithm applied to the weighted automaton A, then*

1. *the weight of a successful path π is unchanged after application of weight pushing:*

$$\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] = \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]). \tag{39}$$

2. *the weighted automaton B is* stochastic, *i.e.*

$$\forall q \in Q, \bigoplus_{e' \in E'[q]} w[e'] = \overline{1}. \tag{40}$$

These two properties of weight pushing are illustrated by Figures 12(a)-(c): the total weight of a successful path is unchanged after pushing; at each state of the weighted automaton of Figure 12(b), the minimum weight of the outgoing transitions is 0, and at at each state of the weighted automaton of Figure 12(c), the weights of outgoing transitions sum to 1.

Weight pushing can also be used to test the equivalence of two subsequential weighted automata [38, 43]. Let A and B be two subsequential weighted automata to which weight pushing can be applied and let A' and B' be the resulting automata after weight pushing. Then, the equivalence of A and B can be tested by applying the standard equivalence algorithm for unweighted automata [1] to A' and B' after considering each pair of (transition label, transition weight) as a single label. The equivalence of two arbitrary weighted automata over the probability semiring can be tested in cubic time using an algorithm [19] based on the standardization technique of Schützenberger [55]. The equivalence of arbitrary weighted automata over the tropical semiring is known to be undecidable [34].

6.4 Minimization

A deterministic weighted automaton is said to be *minimal* if there exists no other deterministic weighted automaton with a smaller number of states and realizing the same function. Two states of a deterministic weighted automaton are said to be *equivalent* if exactly the same set of strings with the same weights label paths from these states to a final state, the final weights being included. Thus, two equivalent states of a deterministic weighted automaton can be merged without affecting the function realized by that automaton. A weighted automaton is minimal when it admits no two distinct equivalent states after any redistribution of the weights along its paths.

There exists a general algorithm for computing a minimal deterministic automaton equivalent to a given weighted automaton [38]. It is thus a generalization of the minimization algorithms for unweighted finite automata. In fact, minimization of both unweighted [39] and weighted finite-state transducers can be viewed as special instances of this algorithm.

The algorithm consists of first applying weight pushing to normalize the distribution of the weights along the paths of the input automaton, and then applying the classical unweighted automata minimization while treating each pair (label, weight) as a single label.

Theorem 8 ([38]). *Let A be a deterministic weighted automaton over a semiring S. Assume that the conditions of application of the weight pushing algorithm hold, then the execution of the following steps:*

1. *weight pushing,*
2. *(unweighted) automata minimization, treating each pair (label, weight) as a single label,*

yield a minimal weighted automaton equivalent to A.

The complexity of automata minimization is linear in the case of acyclic automata $O(|Q| + |E|)$ [53] and in $O(|E| \log |Q|)$ in the general case [1]. Thus, in view of the complexity results given in the previous section, in the case of the tropical semiring, the total complexity of the weighted minimization algorithm is linear in the acyclic case $O(|Q| + |E|)$ and in $O(|E| \log |Q|)$ in the general case.

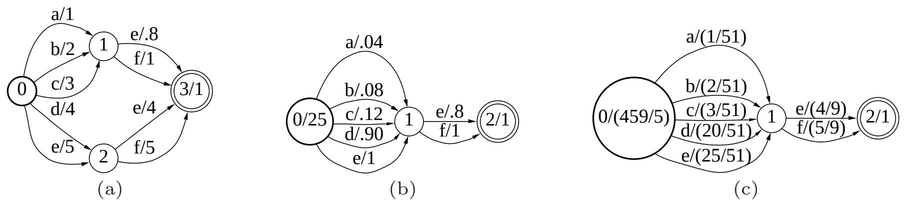


Fig. 13. Minimization of weighted automata. (a) Weighted automaton A' over the probability semiring. (b) Minimal weighted automaton B' equivalent to A'. (c) Minimal weighted automaton C' equivalent to A'.

Figures 12(a), 12(b), and 12(d) illustrate the application of the algorithm in the tropical semiring. The automaton of Figure 12(a) cannot be further minimized using the classical unweighted automata minimization since no two states are equivalent in that machine. After weight pushing, the automaton (Figure 12(b)) has two states (1 and 2) that can be merged by the classical unweighted automata minimization.

Figures 13(a)-(c) illustrate the minimization of an automaton defined over the probability semiring. Unlike the unweighted case, a minimal weighted automaton is not unique, but all minimal weighted automata have the same graph topology, they only differ by the way the weights are distributed along each path. The weighted automata B' and C' are both minimal and equivalent to A'. B' is obtained from A' using the algorithm described above in the probability semiring and it is thus a stochastic weighted automaton in the probability semiring.

For a deterministic weighted automaton, the first operation of the semiring can be arbitrarily chosen without affecting the definition of the function it realizes. This is because, by definition, a deterministic weighted automaton admits at most one path labeled with any given string. Thus, in the algorithm described in Theorem 8, the weight pushing step can be executed in any semiring S' whose multiplicative operation matches that of S. The minimal weighted automaton obtained by pushing the weights in S' is also minimal in S since it can be interpreted as a (deterministic) weighted automaton over S.

In particular, A' can be interpreted as a weighted automaton over the semiring $(\mathbb{R}_+, \max, \times, 0, 1)$. The application of the weighted minimization algorithm to A' in this semiring leads to the minimal weighted automaton C' of Figure 13(c). C' is also a *stochastic* weighted automaton in the sense that, at any state, the maximum weight of all outgoing transitions is one.

This fact leads to several interesting observations. One is related to the complexity of the algorithms. Indeed, we can choose a semiring S' in which the complexity of weight pushing is better than in S. The resulting automaton is still minimal in S and has the additional property of being stochastic in S'. It only differs from the weighted automaton obtained by pushing weights in S in the way weights are distributed along the paths. They can be obtained from each other by application of weight pushing in the appropriate semiring. In the particular case of a weighted automaton over the probability semiring, it may be preferable to use weight pushing in the $(\max, \times)$-semiring since the complexity of the algorithm is then equivalent to that of classical single-source shortest-paths algorithms. The corresponding algorithm is a special instance of the generic shortest-distance algorithm for k-closed semirings presented earlier in the chapter.

Another important point is that the weight pushing algorithm may not be defined in S because the machine is not zero-sum-free or for other reasons. But an alternative semiring S' can sometimes be used to minimize the input weighted automaton.

The results just presented were all related to the minimization of the number of states of a deterministic weighted automaton. The following simple proposition shows that minimizing the number of states coincides with minimizing the number of transitions.

Proposition 9. *Let A be a minimal deterministic weighted automaton, then A has the minimal number of transitions.*

Proof. Let A be a deterministic weighted automaton with the minimal number of transitions. If two distinct states of A were equivalent, they could be merged, thereby strictly reducing the number of its transitions. Thus, A must be a minimal deterministic automaton. Since, minimal deterministic automata have the same topology, in particular the same number of states and transitions, this proves the proposition. □

6.5 Synchronization

The weight pushing algorithm normalizes the way the weights are distributed along the paths. The algorithm presented in this section, *synchronization of weighted transducers*, normalizes instead the way the input and output labels are shifted with respect to each other along the paths. 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, to the extent that is possible.

The following concept helps analyze and describe the domain of application of the algorithm.

Definition 10. *The* delay *of a path π is defined as the difference of length between its output and input labels:*

$$d[\pi] = |o[\pi]| - |i[\pi]|. \tag{41}$$

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 11. *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 $\sigma[\pi]$ defined by:

$$\sigma[\pi] = \begin{cases} \text{suffix of } o[\pi] \text{ of length } |d[\pi]| & \text{if } d[\pi] \geq 0, \\ \text{suffix of } i[\pi] \text{ of length } |d[\pi]| & \text{otherwise.} \end{cases} \tag{42}$$

For any state $q \in Q$, the *string delay at state* q, $s[q]$, is defined by the set of string delays of the paths from an initial state to q:

$$s[q] = \{\sigma[\pi] : \pi \in P(I, q)\}. \tag{43}$$

Lemma 12. *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.The synchronization algorithm described here [42] applies to all weighted transducers with bounded delays. The following is the pseudocode of the algorithm.

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

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 $\mathcal{Q}$ that contains at any time the set of states of T' to examine. At each execution of the loop of lines 3-19, a new state $p' = (q, x, y)$ is extracted from $\mathcal{Q}$ (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 both ϵ if $x\,i[e]$ or $y\,o[e]$ is the empty string, the first symbol of these strings otherwise. The remaining

```
SYNCHRONIZATION($T$)
 1  $F' \leftarrow Q' \leftarrow E' \leftarrow \emptyset$
 2  $\mathcal{Q} \leftarrow i' \leftarrow \{(i, \epsilon, \epsilon) : i \in I\}$
 3  while $\mathcal{Q} \neq \emptyset$ do
 4      $p' = (q, x, y) \leftarrow$ HEAD($\mathcal{Q}$)
 5      DEQUEUE($\mathcal{Q}$)
 6      if ($q \in F$ and $|x| + |y| = 0$) then
 7          $F' \leftarrow F' \cup \{p'\}; \rho'(p') \leftarrow \rho(q)$
 8      elseif ($q \in F$ and $|x| + |y| > 0$) then
 9          $q' \leftarrow (f, \mathrm{cdr}(x), \mathrm{cdr}(y))$
10          $E' \leftarrow E' \uplus (p', \mathrm{car}(x), \mathrm{car}(y), \rho(q), q')$
11          if ($q' \notin Q'$) then
12              $Q' \leftarrow Q' \cup \{q'\}$; ENQUEUE($\mathcal{Q}, q'$)
13      for each $e \in E[q]$ do
14          if ($|x\, i[e]| > 0$ and $|y\, o[e]| > 0$) then
15              $q' \leftarrow (n[e], \mathrm{cdr}(x\, i[e]), \mathrm{cdr}(y\, o[e]))$
16              $E' \leftarrow E' \uplus \{(p', \mathrm{car}(x\, i[e]), \mathrm{car}(y\, o[e]), w[e], q')\}$
17          else $q' \leftarrow (n[e], x\, i[e], y\, o[e])$
18              $E' \leftarrow E' \uplus \{(p', \epsilon, \epsilon, w[e], q')\}$
19          if ($q' \notin Q'$) then
20              $Q' \leftarrow Q' \cup \{q'\}$; ENQUEUE($\mathcal{Q}, q'$)
21  return $T'$
```

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 $\mathcal{Q}$ if it has never been found before (line 18-19). Figures 14(a)-(b) illustrate the synchronization algorithm just presented.

Theorem 13 ([42]). *Let T be a weighted transducer with bounded delays. Then, if we run the synchronization algorithm just described with input T, the algorithm terminates and returns a synchronized transducer T' equivalent to T.*

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:

$$O((|Q| + |E|)(|\Sigma|^{d[T]} + |\Delta|^{d[T]})). \tag{44}$$

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

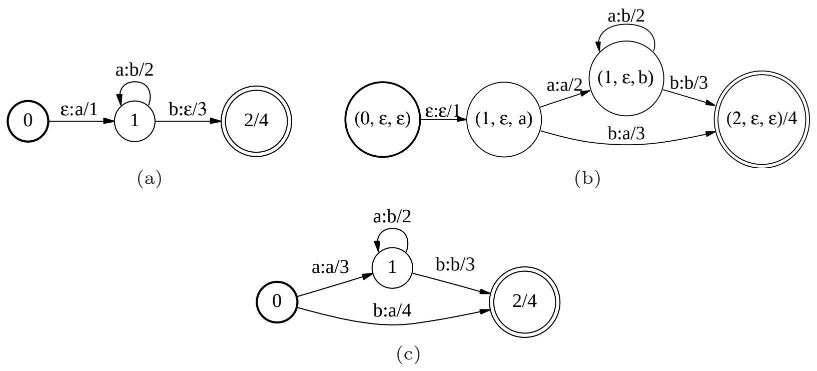


Fig. 14. (a) Weighted transducer T_1 over the tropical semiring. (b) Equivalent synchronized transducer T_2. (c) Synchronized weighted transducer T_3 equivalent to T_1 and T_2 obtained by ϵ-removal from T_2.

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 [27], that is in $O((|\Sigma| + |\Delta|)(|Q| + |E|))$. 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:

$$O((|Q| + |E|)(|\Sigma|^{d[T]} + |\Delta|^{d[T]})). \tag{45}$$

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 $\mathcal{Q}$ 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. 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 S. In particular, the behavior of

Table 2. Properties of several transducer algorithms. The second column indicates the time or space complexity of each algorithm, the third whether or not it admits a natural on-demand computation.

ALGORITHM	COMPLEXITY	ON-DEMAND
Sum	$O(\|T_1\| + \|T_2\|)$	+
Product	$O(\|T_1\| + \|T_2\| + \|F_1\|\|I_2\|)$	+
Closure	$O(\|T\|)$	+
Reversal	$O(\|T\|)$	-
Inversion	$O(\|T\|)$	+
Projection	$O(\|T\|)$	+
Composition	$O(\|T_1\|\|T_2\|)$	+
Intersection	$O(\|A_1\|\|A_2\|)$	+
Completion	$O(\|A\| + \|\Sigma\|)$	+
Complementation[a]	$O(\|A\|)$	+
Difference	$O(\|A_1\|\|A_2\|)$	+
Epsilon-removal[b]	$O(\|Q\|^2 + \|Q\|\|E\|)$	+
Determinization	exponential	+
Minimization[b]	$O(\|E\| \log \|Q\|)$	-
Synchronization[c]	$O((\|Q\| + \|E\|)(\|\Sigma\|^{d[T]} + \|\Delta\|^{d[T]}))$	+

[a] For A deterministic.
[b] For the tropical semiring.
[c] $d(T)$ denotes the maximum delay in an accepting path of T.

the algorithm is identical for two semirings having the same identity elements, such as for example the tropical and log semirings.

The result of the synchronization algorithm may contain ϵ-transitions even if the input contains none. An equivalent weighted transducer with no ϵ-transitions can be computed from T' using the general epsilon-removal algorithm described in a previous section [40]. Figure 14(c) illustrates the result of that algorithm when applied to the synchronized transducer of Figure 14(b). Since epsilon-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 [40] and the total time complexity of epsilon-removal is $O(|Q'|^2 + (T_\oplus + T_\otimes)|Q'||E'|)$. Also, on-demand synchronization can be combined with on-demand epsilon-removal to *directly* create synchronized transducers with no ϵ-transition on-the-fly.

A transducer T is said to be *double-tape unambiguous*, if, when the input and output labels of a transition are treated as single pair label (input label, output label), no two accepting paths have the same label. A by-product of synchronization followed by epsilon-removal is that the resulting transducer is double-tape unambiguous. Note that the definition does not entail any requirement on the weights.

Proposition 14 ([42]). *Let T be a synchronized transducer and assume that T has no ϵ-transition. Then, T is double-tape unambiguous.*

7 Conclusion

Table 2 summarizes some of the essential properties of the algorithms described in this chapter. There are of course many other algorithms related to weighted automata and transducers. But, those presented here constitute some of the core algorithms. Many other algorithms related to weighted automata can be derived either directly from these algorithms or as a combination.

These algorithms are useful in a variety of applications including statistical language modeling [7], parsing [45, 44, 8], phonological rule compilation [29, 30, 49], speech recognition [38, 43, 48], speech synthesis [56, 6], image processing [2], bioinformatics [21, 9], sequence modeling and prediction [18], optical character recognition [14], and more generally any problem related to sequences and probabilistic models defined over sequences [33, 43]. An efficient implementation of these algorithms and several others, including an on-demand implementation when possible, is available from the FSM library (executables only) [47] and the OpenFst library (source and executables) [10].

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley: Reading, MA, 1974.
2. Jürgen Albert and Jarkko Kari. Digital image compression. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of weighted automata*, EATCS Monographs on Theoretical Computer Science. Springer, 2009.
3. Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.
4. Cyril Allauzen and Mehryar Mohri. N-way composition of weighted finite-state transducers. Technical Report TR2007-902, Courant Institute of Mathematical Sciences, New York University, August 2007.
5. Cyril Allauzen and Mehryar Mohri. 3-way composition of weighted finite-state transducers. In *Proceedings of the 13th International Conference on Implementation and Application of Automata (CIAA 2008)*, volume to appear of *Lecture Notes in Computer Science*, San Francisco, California, July 2008. Springer-Verlag, Heidelberg, Germany.
6. Cyril Allauzen, Mehryar Mohri, and Michael Riley. Statistical modeling for unit selection in speech synthesis. In 42*nd Meeting of the Association for Computational Linguistics (ACL 2004), Proceedings of the Conference*, Barcelona, Spain, July 2004.

7. Cyril Allauzen, Mehryar Mohri, and Brian Roark. Generalized algorithms for constructing statistical language models. In *41st Meeting of the Association for Computational Linguistics (ACL 2003), Proceedings of the Conference, Sapporo, Japan*, July 2003.
8. Cyril Allauzen, Mehryar Mohri, and Brian Roark. The design principles and algorithms of a weighted grammar library. *International Journal of Foundations of Computer Science*, 16(3):403–421, 2005.
9. Cyril Allauzen, Mehryar Mohri, and Ameet Talwalkar. Sequence kernels for predicting protein essentiality. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, Helsinki, Finland, July 2008.
10. Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: a general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23, Prague, Czech Republic, July 2007. Springer-Verlag, Heidelberg, Germany.
11. Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: An efficient procedure for deciding functionality and sequentiality. In *Proceedings of LATIN'2000*, volume 1776 of *Lecture Notes in Computer Science*. Springer, 2000.
12. Jean Berstel. *Transductions and Context-Free Languages.* Teubner Studienbucher: Stuttgart, 1979.
13. Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages.* Springer-Verlag: Berlin-New York, 1988.
14. Thomas M. Breuel. The OCRopus open source OCR system. In *Proceedings of IS&T/SPIE 20th Annual Symposium*, 2008.
15. Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoretical Computer Science*, 5:325–338, 1977.
16. Christian Choffrut. *Contributions à l'étude de quelques familles remarquables de fonctions rationnelles.* PhD thesis, (thèse de doctorat d'Etat), Université Paris 7, LITP: Paris, France, 1978.
17. Thomas H. Cormen, Charles E. Leiserson, and Ronald E. Rivest. *Introduction to Algorithms.* The MIT Press: Cambridge, MA, 1992.
18. Corinna Cortes, Patrick Haffner, and Mehryar Mohri. A machine learning framework for spoken-dialog classification. In Larry Rabiner and Fred Juang, editors, *Handbook on Speech Processing and Speech Communication, Part E: Speech recognition*. Springer-Verlag, Heidelberg, Germany, 2008.
19. Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. L_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(4):761–780, 2007.
20. Manfred Droste and Werner Kuich. Semirings and formal power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of weighted automata*, EATCS Monographs on Theoretical Computer Science. Springer, 2009.
21. Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, Cambridge UK, 1998.
22. Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

23. Zoltán Ésik and Werner Kuich. Modern automata theory. `www.dmg.tuwien.ac.at/kuich`.
24. Zoltán Ésik and Werner Kuich. Rationally additive semirings. *Journal of Universal Computer Science*, 8:173–183, 2002.
25. Robert W. Floyd. Algorithm 97 (SHORTEST PATH). *Communications of the ACM*, 18, 1968.
26. Martin Goldstern. Vervollständigung von halbringen. Master's thesis, Technische Universität Wien, 1985.
27. 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.
28. Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.
29. Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3), 1994.
30. Lauri Karttunen. The replace operator. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics, 1995. Distributed by Morgan Kaufmann Publishers, San Francisco, California.
31. Daniel Kirsten. A Burnside approach to the termination of Mohri's algorithm for polynomially ambiguous min-plus-automata. In *R.A.I.R.O. - Informatique Théorique et Applications, special issue on Journées Montoises d'Informatique Théorique 2006 (JM'06)*, to appear in 2008.
32. Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.
33. Kevin Knight and Jonathan May. Applications of weighted automata in natural language processing. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of weighted automata*, EATCS Monographs on Theoretical Computer Science. Springer, 2009.
34. Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Journal of Algebra and Computation*, 4, 1994.
35. Werner Kuich. *Handbook of formal languages, vol. 1: word, language, grammar*, chapter Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata, pages 609–677. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
36. Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin-New York, 1986.
37. Daniel J. Lehmann. Algebraic structures for transitive closures. *Theoretical Computer Science*, 4:59–76, 1977.
38. Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:2, 1997.
39. Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, March 2000.
40. 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.
41. Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

42. Mehryar Mohri. Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(6):957–982, 2003.
43. Mehryar Mohri. Statistical natural language processing. In M. Lothaire, editor, *Applied Combinatorics on Words*. Cambridge University Press, 2005.
44. Mehryar Mohri and Mark-Jan Nederhof. Regular approximation of context-free grammars through transformation. In *Robustness in Language and Speech Technology*, pages 153–163. Kluwer Academic Publishers, The Netherlands, 2001.
45. Mehryar Mohri and Fernando C. N. Pereira. Dynamic Compilation of Weighted Context-Free Grammars. In 36*th Meeting of the Association for Computational Linguistics (ACL '98), Proceedings of the Conference, Montréal, Québec, Canada*, pages 891–897, 1998.
46. 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, 1996. John Wiley and Sons, Chichester.
47. 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.
48. Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In Larry Rabiner and Fred Juang, editors, *Handbook on speech processing and speech communication, Part E: Speech recognition*. Springer-Verlag, Heidelberg, Germany, 2008.
49. 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*, Santa Cruz, California, 1996.
50. Fernando C. N. Pereira and Michael D. Riley. Speech Recognition by Composition of Weighted Finite Automata. In *Finite-State Language Processing*, pages 431–453. MIT Press, 1997.
51. Dominique Perrin. Finite automata. In J. Van Leuwen, editor, *Handbook of theoretical computer science, volume B: Formal models and semantics*, pages 1–57. Elsevier, Amsterdam, 1990.
52. Ion Petre and Arto Salomaa. Algebraic systems and pushdown automata. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of weighted automata*, EATCS Monographs on Theoretical Computer Science. Springer, 2009.
53. Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189, 1992.
54. Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag: New York, 1978.
55. Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4, 1961.
56. Richard Sproat. A finite-state architecture for tokenization and grapheme-to-phoneme conversion in multilingual text analysis. In *Proceedings of the ACL SIGDAT Workshop, Dublin, Ireland*. ACL, 1995.
57. Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.
58. Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Information and Computation*, 118(2):327–340, 1995.

Index

ϵ-closure, 21

algorithm, 37
  optimization, 21

cancellative operation, 25
closure, 6, 7, 11
  of a matrix, 7
  transitive, 8
complementation, 20
completion, 20
composition, 14
  N-way, 18
concatenation, 11
connection, 4, 15, 19

determinization, 25
difference, 19

empty string (ϵ), 2
Epsilon-Removal, 22
epsilon-removal, 21
  reverse, 25
equivalent, 31

finite-state transducer, 2, 4
finite automaton, 4

Gen-All-Pairs, 8
Gen-Single-Source, 10

Hadamard product, 18

intersection, 18

inversion, 13

Kleene-closure, *see* closure

minimal, 31
minimization, 31
mirror image, 2
monoid, 2

on-demand construction, 12
on-the-fly construction, 12
operation
  binary, 14
  closure, 6, 11
  rational, 11
  unary, 13

path
  accepting, 3
  delay, 33
  successful, 3
product, 11
projection, 13
  input, 13
  left, 13
  output, 13
  right, 13

rational power series, 1
reversal, 13
reweighting, 29

semiring, 2
  k-closed, 9

Boolean, 2, 6
cancellative, 25
closed, 5
commutative, 2
complete, 5
Conway, 6, 7
divisible, 25
idempotent, 2
lattice, 6
locally closed, 9
log, 2, 6
non-idempotent, 6
probability, 2, 6
tropical, 2, 6
weakly divisible, 25
shortest-distance, 6
algorithm, 5
all-pairs, 6, 8
single-source, 9, 10
problem
all-pairs, 5, 7
single-source, 9
starsemiring, 6
complete, 6
state
final, 3
initial, 3
non-accessible, 4
non-coaccessible, 4
sibling, 27
twin, 27
useless, 4
stochastic, 30, 32
string
delay, 34
length of, 2
mirror image of, 2
strongly connected component, 8
decomposition into, 8
sum, 11
SYNCHRONIZATION, 35
synchronization, 33

transducer, *see* weighted transducer
transposition, 13
trimming, 4, 15, 19
twins property, 27, 28

union, 11
unweighted automaton, *see* finite automaton
unweighted transducer, *see* finite-state transducer

Viterbi approximation, 2

weight
final, 3
initial, 3, 4
weighted automaton
determinizable, 26, 27
subsequentiable, 27
weighted subset, 25
WEIGHTED-COMPOSITION, 16
WEIGHTED-DETERMINIZATION, 26
weighted automaton, 1, 2, 4
deterministic, 25
polynomially ambiguous, 28
subsequential, 25
weighted finite-state transducer, *see* weighted transducer
weighted transducer, 1–3
complete, 20
deterministic, 4
determinizable, 24
double-tape unambiguous, 37
inverse, 3
regulated, 3
sequential, 4
synchronized, 34
trim, 4
unambiguous, 4
weight function, *see* weight
weight pushing, 29