3-Way Composition of Weighted Finite-State Transducers

Cyril Allauzen\textsuperscript{1,*} and Mehryar Mohri\textsuperscript{1,2}

\textsuperscript{1} Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012.
\textsuperscript{2} Google Research, 76 Ninth Avenue, New York, NY 10011.

Abstract. Composition of weighted transducers is a fundamental algorithm used in many applications, including for computing complex edit-distances between automata, or string kernels in machine learning, or to combine different components of a speech recognition, speech synthesis, or information extraction system. We present a generalization of the composition of weighted transducers, 3-way composition, which is dramatically faster in practice than the standard composition algorithm when combining more than two transducers. The worst-case complexity of our algorithm for composing three transducers $T_1$, $T_2$, and $T_3$ resulting in $T$, is $O(|T| Q \cdot \min(d(T_1) d(T_3), d(T_2)) + |T| E)$, where $|\cdot|_Q$ denotes the number of states, $|\cdot|_E$ the number of transitions, and $d(\cdot)$ the maximum out-degree. As in regular composition, the use of perfect hashing requires a pre-processing step with linear-time expected complexity in the size of the input transducers. In many cases, this approach significantly improves on the complexity of standard composition. Our algorithm also leads to a dramatically faster composition in practice. Furthermore, standard composition can be obtained as a special case of our algorithm. We report the results of several experiments demonstrating this improvement. These theoretical and empirical improvements significantly enhance performance in the applications already mentioned.

1 Introduction

Weighted finite-state transducers are widely used in text, speech, and image processing applications and other related areas such as information extraction [8, 10, 12, 11, 4]. They are finite automata in which each transition is augmented with an output label and some weight, in addition to the familiar (input) label [14, 5, 7]. The weights may represent probabilities, log-likelihoods, or they may be some other costs used to rank alternatives. They are, more generally, elements of a semiring [7].

Weighted transducers are used to represent models derived from large data sets using various statistical learning techniques such as pronunciation dictionaries, statistical grammars, string kernels, or complex edit-distance models [11, 6, 2, 3]. These models can be combined to create complex systems such as a speech recognition or information extraction system using a fundamental transducer algorithm, composition of weighted

\* This author’s current address is: Google Research, 76 Ninth Avenue, New York, NY 10011.
transducers [12, 11]. Weighted composition is a generalization of the composition algorithm for unweighted finite-state transducers which consists of matching the output label of the transitions of one transducer with the input label of the transitions of another transducer. The weighted case is however more complex and requires the introduction of an $\epsilon$-filter to avoid the creation of redundant $\epsilon$-paths and preserve the correct path multiplicity [12, 11]. The result is a new weighted transducer representing the relational composition of the two transducers.

Composition is widely used in computational biology, text and speech, and machine learning applications. In many of these applications, the transducers used are quite large, they may have as many as several hundred million states or transitions. A critical problem is thus to devise efficient algorithms for combining them. This paper presents a generalization of the composition of weighted transducer, 3-way composition, that is dramatically faster than the standard composition algorithm when combining more than two transducers. The complexity of composing three transducer $T_1$, $T_2$, and $T_3$, with the standard composition algorithm is $O(|T_1||T_2||T_3|)$ [12, 11]. Using perfect hashing, the worst-case complexity of computing $T = (T_1 \circ T_2) \circ T_3$ using standard composition is

$$O(|T|Q \min(d(T_3), d(T_1 \circ T_2)) + |T|_E + |T_1 \circ T_2|Q \min(d(T_1), d(T_2)) + |T_1 \circ T_2|_E),$$

which may be prohibitive in some cases even when the resulting transducer $T$ is not large but the intermediate transducer $T_1 \circ T_2$ is. Instead, the worst-case complexity of our algorithm is

$$O(|T|Q \min(d(T_1), d(T_3)) + d(T_2) + |T|_E).$$

In both cases, the use of perfect hashing requires a pre-processing step with linear-time expected complexity in the size of the input transducers.

Our algorithm also leads to a dramatically faster computation of the result of composition in practice. We report the results of several experiments demonstrating this improvement. These theoretical and empirical improvements significantly enhance performance in a series of applications: string kernel-based algorithms in machine learning, the computation of complex edit-distances between automata, speech recognition and speech synthesis, and information extraction. Furthermore, as we shall see later, standard composition can be obtained as a special case of 3-way composition.

The main technical difficulty in the design of our algorithm is the definition of a filter to deal with a path multiplicity problem that arises in the presence of the empty string $\epsilon$ in the composition of three transducers. This problem, which we shall describe in detail, leads to a word combinatorial problem [13]. We will present two solutions for this problem: one requiring two $\epsilon$-filters and a generalization of the $\epsilon$-filters used for standard composition [12, 11]; and another direct and symmetric solution where a single filter is needed. Remarkably, this 3-way filter can be encoded as a finite automaton and painlessly integrated in our 3-way composition.

The remainder of the paper is structured as follows. Some preliminary definitions and terminology are introduced in the next section (Section 2). Section 3 describes our 3-way algorithm in the $\epsilon$-free case. The word combinatorial problem of $\epsilon$-path multiplicity and our solutions are presented in detail Section 4. Section 5 reports the results of experiments using the 3-way algorithm and compares them with the standard composition.
2 Preliminaries

This section gives the standard definition and specifies the notation used for weighted transducers.

Finite-state transducers are finite automata in which each transition is augmented with an output label in addition to the familiar input label [1, 5]. 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 [14, 7]. The weights are elements of a semiring, that is a ring that may lack negation [7]. Some familiar semirings are the tropical semiring \((\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)\) related to classical shortest-paths algorithms, and the probability semiring \((\mathbb{R}, +, \cdot, 0, 1)\). A semiring is idempotent if for all \(a \in K\), \(a \circ a = a\). It is commutative when \(\otimes\) is commutative.

We will assume in this paper that the semiring used is commutative, which is a necessary condition for composition to be an efficient algorithm [10]. The following gives a formal definition of weighted transducers.

**Definition 1.** A weighted finite-state transducer \(T\) over \((K, \oplus, \cdot, 0, 1)\) is an 8-tuple \(T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)\) where \(\Sigma\) is the finite input alphabet of the transducer, \(\Delta\) is the finite output alphabet, \(Q\) is a finite set of states, \(I \subseteq Q\) the set of initial states, \(F \subseteq Q\) the set of final states, \(E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times K \times Q\) a finite set of transitions, \(\lambda : I \rightarrow K\) the initial weight function, and \(\rho : F \rightarrow K\) the final weight function mapping \(F\) to \(K\).

The weight of a path \(\pi\) is obtained by multiplying the weights of its constituent transitions using the multiplication rule of the semiring and is denoted by \(w[\pi]\). The weight of a pair of input and output strings \((x, y)\) is obtained by \(\oplus\)-summing the weights of the paths labeled with \((x, y)\) from an initial state to a final state.

For a path \(\pi\), we denote by \(p[\pi]\) its origin state and by \(n[\pi]\) its destination state. We also denote by \(P(I, x, y, F)\) the set of paths from the initial states \(I\) to the final states \(F\) labeled with input string \(x\) and output string \(y\). A transducer \(T\) is regulated if the output weight associated by \(T\) to any pair of strings \((x, y)\):

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

(3)

is well-defined and in \(K\). \(T(x, y) = \emptyset\) when \(P(I, x, y, F) = \emptyset\). If for all \(q \in Q\)
\[
\bigoplus_{\pi \in P(q, \epsilon, \epsilon)\, w[\pi]} \in K,
\]
then \(T\) is regulated. In particular, when \(T\) does not admit any \(\epsilon\)-cycle, it is regulated. The weighted transducers we will be considering in this paper will be regulated. Figure 1(a) shows an example.

The composition of two weighted transducers \(T_1\) and \(T_2\) with matching input and output alphabets \(\Sigma\), is a weighted transducer denoted by \(T_1 \circ T_2\) when the sum:

\[
(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \Sigma^*} T_1(x, z) \otimes T_2(z, y)
\]

(4)

is well-defined and in \(K\) for all \(x, y \in \Sigma^*\) [14, 7]. Weighted automata can be defined as weighted transducers \(A\) with identical input and output labels, for any transition. Thus,
transitions of the transducer with the highest out-degree has an expected complexity in \( O(n^2) \). Thus, first putting a transducer with the highest out-degree leads to a worst-case complexity of \( O(n^2 \times m^2) \). The worst-case complexity of that computation is:

\[
O(|T_1| \times |T_2|) = O(n^2 \times m^2).
\]

Figure 1 illustrates the algorithm. 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: \( O(|T_1| \times |T_2|) \). However, using perfect hashing on the input transducer with the highest out-degree leads to a worst-case complexity of \( O(|T_1| \times |T_2|) \). The pre-processing step required for hashing the transitions of the transducer with the highest out-degree has an expected complexity in \( O(|T_1|) \) if \( d(T_1) > d(T_2) \) and \( O(|T_2|) \) otherwise.

The main problem with the standard composition algorithm is the following. Assume that one wishes to compute \( T_1 \circ T_2 \circ T_3 \), say for example by proceeding left to right. Thus, first \( T_1 \) and \( T_2 \) are composed to compute \( T_1 \circ T_2 \) and then the result is composed with \( T_3 \). The worst-case complexity of that computation is:

\[
O(|T_1| \times |T_2|) = O(n^2 \times m^2).
\]

only 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.

### 3 Epsilon-Free Composition

#### 3.1 Standard Composition

Let us start with a brief description of the standard composition algorithm for weighted transducers [12, 11]. States in the composition \( T_1 \circ T_2 \) of two weighted transducers \( T_1 \) and \( T_2 \) are identified with pairs of a state of \( T_1 \) and a state of \( T_2 \). Leaving aside transitions with \( \epsilon \) inputs or outputs, the following rule specifies how to compute a transition of \( T_1 \circ T_2 \) from appropriate transitions of \( T_1 \) and \( T_2 \):

\[
(q_1, a, b, w_1, q_2) \quad \text{and} \quad (q_1', b, c, w_2, q_2') \quad \Rightarrow \quad ((q_1, q_1'), a, c, w_1 \otimes w_2, (q_2, q_2')).
\]

(5)

Figure 2 illustrates the algorithm. 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: \( O(|T_1| \times |T_2|) \). However, using perfect hashing on the input transducer with the highest out-degree leads to a worst-case complexity of \( O(|T_1| \times |T_2|) \). The pre-processing step required for hashing the transitions of the transducer with the highest out-degree has an expected complexity in \( O(|T_1|) \) if \( d(T_1) > d(T_2) \) and \( O(|T_2|) \) otherwise.

The main problem with the standard composition algorithm is the following. Assume that one wishes to compute \( T_1 \circ T_2 \circ T_3 \), say for example by proceeding left to right. Thus, first \( T_1 \) and \( T_2 \) are composed to compute \( T_1 \circ T_2 \) and then the result is composed with \( T_3 \). The worst-case complexity of that computation is:

\[
O(|T_1| \times |T_2|) \times O(d(T_1) \times d(T_2)) = O(|T_1| \times |T_2| \times d(T_1) \times d(T_2)).
\]

(6)
Fig. 2. Example of transducer composition. (a) Weighted transducer $T_1$ and (b) Weighted transducer $T_2$ over the probability semiring $(\mathbb{R}, +, \cdot, 0, 1)$. (c) Result of the composition of $T_1$ and $T_2$.

But, in many cases, computing $T_1 \circ T_2$ creates a very large number of transitions that may never match any transition of $T_3$. For example, $T_2$ may represent a complex edit-distance transducer, allowing all possible insertions, deletions, substitutions and perhaps other operations such as transpositions or more complex edits in $T_1$ all with different costs. Even when $T_1$ is a simple non-deterministic finite automaton with $\epsilon$-transitions, which is often the case in the applications already mentioned, $T_1 \circ T_2$ will then have a very large number of paths, most of which will not match those of the non-deterministic automaton $T_3$. In other applications in speech recognition, or for the computation of kernels in machine learning, the central transducer $T_2$ could be far more complex and the set of transitions or paths of $T_1 \circ T_2$ not matching those of $T_3$ could be even larger.

3.2 3-Way Composition

The key idea behind our algorithm is precisely to avoid creating these unnecessary transitions by directly constructing $T_1 \circ T_2 \circ T_3$, which we refer to as a 3-way composition. Thus, our algorithm does not include the intermediate step of creating $T_1 \circ T_2$ or $T_2 \circ T_3$. To do so, we can proceed following a lateral or sideways strategy: for each transition $e_1$ in $T_1$ and $e_3$ in $T_3$, we search for matching transitions in $T_2$.

The pseudocode of the algorithm in the $\epsilon$-free case is given below. The algorithm computes $T$, the result of the composition $T_1 \circ T_2 \circ T_3$. It uses a queue $S$ containing the set of pairs of states yet to be examined. The queue discipline of $S$ can be arbitrarily chosen and does not affect the termination of the algorithm. Using a FIFO or LIFO discipline, the queue operations can be performed in constant time. We can pre-process the transducer $T_2$ in expected linear time $O(|T_2|E)$ by using perfect hashing so that the transitions $G$ (line 13) can be found in worst-case linear time $O(|G|)$. Thus, the worst-case running time complexity of the 3-way composition algorithm is in $O(|T|Qd(T_1)d(T_3) + |T|E)$, where $T$ is transducer returned by the algorithm.

Alternatively, depending on the size of the three transducers, it may be advantageous to direct the 3-way composition from the center, i.e., ask for each transition $e_2$ in $T_2$ if there are matching transitions $e_1$ in $T_1$ and $e_3$ in $T_3$. We refer to this as the central strategy for our 3-way composition algorithm. Pre-processing the transducers $T_1$ and $T_3$ and creating hash tables for the transitions leaving each state (the expected complexity of this pre-processing being $O(|T_1|E + |T_3|E)$), this strategy leads to a worst-case running
time complexity of $O(|T|Qd(T_2) + |T|E)$. The lateral and central strategies can be combined by using, at a state $(q_1, q_2, q_3)$, the lateral strategy if $|E[q_1]| \cdot |E[q_3]| \leq |E[q_2]|$ and the central strategy otherwise. The algorithm leads to a natural lazy or on-demand implementation in which the transitions of the resulting transducer $T$ are generated only as needed by other operations on $T$. The standard composition coincides with the 3-way algorithm when using the central strategy with either $T_1$ or $T_2$ equal to the identity transducer.

3-WAY-COMPOSITION($T_1, T_2, T_3$)
1  $Q \leftarrow I_1 \times I_2 \times I_3$
2  $S \leftarrow I_1 \times I_2 \times I_3$
3  while $S \neq \emptyset$ do
4    $(q_1, q_2, q_3) \leftarrow \text{HEAD}(S)$
5    DEQUEUE($S$)
6    if $(q_1, q_2, q_3) \in I_1 \times I_2 \times I_3$ then
7      $I \leftarrow I \cup \{(q_1, q_2, q_3)\}$
8      $\lambda(q_1, q_2, q_3) \leftarrow \lambda_1(q_1) \circ \lambda_2(q_2) \circ \lambda_3(q_3)$
9    else
10       $(q_1, q_2, q_3) \in F_1 \times F_2 \times F_3$ then
11          $F \leftarrow F \cup \{(q_1, q_2, q_3)\}$
12          $\rho(q_1, q_2, q_3) \leftarrow \rho_1(q_1) \circ \rho_2(q_2) \circ \rho_3(q_3)$
13    for each $(e_1, e_3) \in E[q_1] \times E[q_3]$ do
14      $G \leftarrow \{e \in E[q_2] : i[e] = o[e_1] \land o[e] = i[e_3]\}$
15      for each $e_2 \in G$ do
16        if $(n[e_1], n[e_2], n[e_3]) \notin Q$ then
17          $Q \leftarrow Q \cup \{(n[e_1], n[e_2], n[e_3])\}$
18          ENQUEUE($S, (n[e_1], n[e_2], n[e_3])$)
19      $E \leftarrow E \cup \{(q_1, q_2, q_3), i[e_1], o[e_3], w[e_1] \otimes w[e_2] \otimes w[e_3], (n[e_1], n[e_2], n[e_3])\}$
20  return $T$

4 Epsilon filtering

The algorithm described thus far cannot be readily used in most cases found in practice. In general, a transducer $T_1$ may have transitions with output label $\epsilon$ and $T_2$ transitions with input $\epsilon$. A straightforward generalization of the $\epsilon$-free case would generate redundant $\epsilon$-paths and, in the case of non-idempotent semirings, would lead to an incorrect result, even just for composing two transducers. The weight of two matching $\epsilon$-paths of the original transducers would be counted as many times as the number of redundant $\epsilon$-paths generated in the result, instead of one. Thus, a crucial component of our algorithm consists of coping with this problem.

Figure 3(a) illustrates the problem just mentioned in the simpler case of two transducers. To match $\epsilon$-paths leaving $q_1$ and those leaving $q_2$, a generalization of the $\epsilon$-free composition can make the following moves: (1) first move forward on a transition of $q_1$ with output $\epsilon$, or even a path with output $\epsilon$, and stay at the same state $q_2$ in $T_2$, with the hope of later finding a transition whose output label is some label $a \neq \epsilon$ matching a transition of $q_2$ with the same input label; (2) proceed similarly by following a transition or path leaving $q_2$ with input label $\epsilon$ while staying at the same state $q_1$ in $T_1$; or, (3) match a transition of $q_1$ with output label $\epsilon$ with a transition of $q_2$ with input label $\epsilon$. 


Proposition 1. Let $M$ be the transducer of Figure 3(b). $M$ allows a unique path between any two states $(p, q)$ and $(r, s)$, with $(p, q) \preceq (r, s)$.

Proof. Let $a$ denote $(\epsilon_1: \epsilon_1)$, $b$ denote $(\epsilon_2: \epsilon_2)$, $c$ denote $(\epsilon_2: \epsilon_1)$, and let $x$ stand for any $(x:x)$, with $x \in \Sigma$. The following sequences must be disallowed by a shortest-path filter with matching transitions first: $ab, ba, ac, bc$. This is because, from any state, instead of the moves $ab$ or $ba$, the matching or diagonal transition $c$ can be taken. Similarly, instead of $ac$ or $bc$, $ca$ and $cb$ can be taken for an earlier match. Conversely, it is clear from the grid or an immediate recursion that a filter disallowing these sequences accepts a unique path between two connected states of the grid.

Let $L$ be the set of sequences over $\sigma = \{a, b, c, x\}$ that contain one of the disallowed sequence just mentioned as a substring that is $L = \sigma^*(ab + ba + ac + bc)\sigma^*$. Then $\overline{L}$ represents exactly the set of paths allowed by that filter and is thus a regular language. Let $A$ be an automaton representing $L$ (Figure 4(a)). An automaton representing $\overline{L}$ can
be constructed from $A$ by determinization and complementation (Figures 4(a)-(c)). The resulting automaton $C$ is equivalent to the transducer $M$ after removal of the state 3, which does not admit a path to a final state. □

Thus, to compose two transducers $T_1$ and $T_2$ with $\epsilon$-transitions, it suffices to compute $T_1 \circ M \circ T_2$, using the rules of composition in the $\epsilon$-free case.

The problem of avoiding the creation of redundant $\epsilon$-paths is more complex in 3-way composition since the $\epsilon$-transitions of all three transducers must be taken into account. We describe two solutions for this problem, one based on two filters, another based on a single filter.

4.1 2-way $\epsilon$-Filters.

One way to deal with this problem is to use the 2-way filter $M$, by first dealing with matching $\epsilon$-paths in $U = (T_1 \circ T_2)$, and then $U \circ T_3$. However, in 3-way composition, it is possible to remain at the same state of $T_1$ and the same state of $T_2$, and move on an $\epsilon$-transition of $T_3$, which previously was not an option. This corresponds to staying at the same state of $U$, while moving on a transition of $T_3$ with input $\epsilon$. To account for this move, we introduce a new symbol $\epsilon_0$ matching $\epsilon_1$ in $T_3$. But, we must also ensure the existence of a self-loop with output label $\epsilon_0$ at all states of $U$. To do so, we augment the filter $M$ with self-loops $(\epsilon_1: \epsilon_0)$ and the transducer $T_2$ with self-loops $(\epsilon_0: \epsilon_1)$ (see Figure 5(b)). Figure 5(d) shows the resulting filter transducer $M_1$. From Figures 5(a)-(c), it is clear that $\tilde{T}_1 \circ M_1 \circ \tilde{T}_2$ will have precisely a self-loop labeled with $(\epsilon_1: \epsilon_1)$ at all states.

In the same way, we must allow for moving forward on a transition of $T_1$ with output $\epsilon$, that is consuming $\epsilon_2$, while remaining at the same states of $T_2$ and $T_3$. To do so, we introduce again a new symbol $\epsilon_0$ this time only relevant for matching $T_2$ with $T_3$, add self-loops $(\epsilon_2: \epsilon_0)$ to $T_2$, and augment the filter $M$ by adding a transition labeled with $(\epsilon_0: \epsilon_2)$ (resp. $(\epsilon_0: \epsilon_1)$) wherever there used to be one labeled with $(\epsilon_2: \epsilon_2)$ (resp. $(\epsilon_2: \epsilon_1)$). Figure 5(e) shows the resulting filter transducer $M_2$.

Thus, the composition $\tilde{T}_1 \circ M_1 \circ \tilde{T}_2 \circ M_2 \circ T_3$ ensures the uniqueness of matching $\epsilon$-paths. In practice, the modifications of the transducers $T_1$, $T_2$, and $T_3$ to generate $\tilde{T}_1$, $\tilde{T}_2$, and $T_3$, as well as the filters $M_1$ and $M_2$ can be directly simulated or encoded in the
3-way composition algorithm for greater efficiency. The states in $T$ become quintuples $(q_1, q_2, q_3, f_1, f_2)$ with $f_1$ and $f_2$ are states of the filters $M_1$ and $M_2$. The introduction of self-loops and marking of $\epsilon$-s can be simulated (line 12-13) and the filter states $f_1$ and $f_2$ taken into account to compute the set $G$ of the transition matches allowed (line 13).

Note that while 3-way composition is symmetric, the analysis of $\epsilon$-paths just presented is left-to-right and the filters $M_1$ and $M_2$ are not symmetric. In fact, we could similarly define right-to-left filters $M_1'$ and $M_2'$. The advantage of the filters presented in this section is however that they can help modify easily an existing implementation of composition into 3-way composition. The filters needed for the 3-way case are also straightforward generalizations of the $\epsilon$-filter used in standard composition.

4.2 3-way $\epsilon$-Filter.

There exists however a direct and symmetric method for dealing with $\epsilon$-paths in 3-way composition. Remarkably, this can be done using a single filter automaton whose labels are 3-dimensional vectors. Figure 6 shows a filter $W$ that can be used for that purpose. Each transition is labeled with a triplet. The $i$th element of the triplet corresponding to the move on the $i$th transducer. 0 indicates staying at the same state or not moving, 1 that a move is made reading an $\epsilon$-transition, and $x$ a move along a matching transition with a non-empty symbol (i.e., non-$\epsilon$ output in $T_1$, non-$\epsilon$ input or output in $T_2$ and non-$\epsilon$ input in $T_3$).

Matching $\epsilon$-paths now correspond to a three-dimensional grid, which leads to a more complex word combinatorics problem. As in the two-dimensional case, $(p, q, r) \preceq (s, t, u)$ indicates that $(s, t, u)$ can be reached from $(p, q, r)$ in the grid. Several filters are possible, here we will again favor the matching of $\epsilon$-transitions (i.e. the diagonals on the grid).

**Proposition 2.** The filter automaton $W$ allows a unique path between any two states $(p, q, r)$ and $(s, t, u)$ of a three-dimensional grid, with $(p, q, r) \preceq (s, t, u)$. 
Proof. Let $\mathcal{M}$ and $\mathcal{X}$ be the defined by $\mathcal{M} = \{(m_1, m_2, m_3) : m_1, m_2, m_3 \in \{0, 1\}\}$ and $\mathcal{X} = \{(x, x, m), (m, x, x) : m \in \{0, 1\}\}$. A sequence of moves corresponding to a matching $\epsilon$-path is thus an element of $(\mathcal{M} \cup \mathcal{X})^*$. Two sequences $\pi_1$ and $\pi_2$ are equivalent if they consume the same sequence of transitions on each of the three transducers, for example $(0, x, x)(1, 1, 0)$ is equivalent to $(1, x,)(0, 1, 0)$. For each set of equivalent move sequences between two states $(p, q, r)$ and $(s, t, u)$, we must preserve a unique sequence representative of that set. We now define the unique corresponding representative $\#\pi$ of each sequence $\pi \in (\mathcal{M} \cup \mathcal{X})^*$. In all cases, $\#\pi$ will be the sequence where the 1-moves and the x-moves are taken as early as possible.

1. Assume that $\pi \in \mathcal{M}^*$ and let $n_i$ be the number of occurrences of 1 as the $i$th element in a triplets defining $\pi$. By symmetry, we can assume without loss of generality that $n_1 \leq n_2 \leq n_3$. We define $\#\pi$ as $(1, 1, 1)^{n_1}(0, 1, 1)^{n_2-n_1}(0, 0, 1)^{n_3-n_2}$, that is the sequence where the 1-moves are taken as early as possible.

2. Otherwise, $\pi$ can be decomposed as $\pi = \mu_1 \chi_1 \mu_2 \chi_2 \cdots \mu_k \chi_k \mu_{k+1}$ with $k \geq 1$, $\mu_i \in \mathcal{M}^*$ and $\chi_i \in \mathcal{X}$. $\#\pi$ is then defined by induction on $k$. By symmetry, we can assume that $\chi_1 = (x, x, m)$ with $m \in \{0, 1\}$. Let $\pi'$ be such that $\pi = \mu_1 \chi_1 \pi'$, and $n_i$ be the number of times 1 appears as ith element in a triplet $\mu_1$, and let $n'_i$ the number of times 1 is found as third element in a triplet reading $\chi_1 \pi'$ from left to right before seeing an $x$.

(a) If $n_3 \leq \max(n_1, n_2)$, let $n = \min(n'_3, \max(n_1, n_2) - n_3)$. We can then obtain $\chi_1^i \pi''$ by replacing the $n$ first 1’s that appears in $\chi_1 \pi'$ as third element of a triplet by 0’s. Let $\mu'_1 = \mu_1(0, 0, 1)^n$. We then have that $\pi$ is equivalent to $\mu'_1 \chi_1^i \pi''$. By induction, we can compute $\#\pi''$ and $\#\pi'$ and define $\#\pi$ as $\mu'_1 \chi_1 \#\pi''$.

(b) If $n_3 > \max(n_1, n_2)$, we define $n$ as $n_3 - \max(n_1, n_2)$ if $\chi_1 = (x, x, 1)$ and $n_3 - \max(n_1, n_2) - 1$ if $\chi_1 = (x, x, 0)$. Let $\mu'_1$ be $(1, 1, 1)^{n_1}(0, 1, 1)^{n_2-n_1}$ if $n_1 < n_2$ and $(1, 1, 1)^{n_1}(0, 0, 1)^{n_3-n_2}$ otherwise. We can then define $\#\pi$ as $\mu'_1(x, x, 1)(0, 0, 1)^{n\pi'}$. 

Fig. 6. 3-way matching $\epsilon$-filter $W$. 
A key property of $\pi$ is that it can be characterized by a small set of forbidden sequences. Indeed, observe that the following rules apply:

1. In two consecutive triplets, for $i \in [1,3]$, 0 in the $i$th machine of the first triplet cannot be followed by 1 in the second. Indeed, as in the 2-way case, if we stay at a state, then we must remain at that state until a match with a non-empty symbol is made (this correspond to cases 1 and 2(a) of the definition of $\pi$).

2. Two 0s in adjacent transducers ($T_1$ and $T_2$, or $T_2$ and $T_3$), cannot become both $x$s unless all components become $x$s; For example, the sequence $(0, 0, 1)(x, x, 1)$ is disallowed since instead $(x, x, 1)(0, 0, 1)$ with an earlier match can be followed. Similarly, the sequence $(0, 0, 1)(x, x, 0)$ is disallowed since instead the single and shorter move $(x, x, 1)$ can be taken (this correspond to case 2(b) of the definition).

3. The triplet $(0, 0, 0)$ is always forbidden since it corresponds to remaining at the same state in all three transducers.

Conversely, we observe that with our definition of $\bar{\pi}$, these conditions are also sufficient. Thus, a filter can be obtained by taking the complement of an automaton accepting exactly the sequences of forbidden substrings just described. The resulting deterministic and minimal automaton is the filter $W$ shown in Figure 6. Observe that each state of $W$ has a transition labeled by $(x, x, x)$ going to the initial state 0, this corresponds to resetting the filter at the end of a matching $\epsilon$-path.

The filter $W$ is used as follows. A triplet state $(q_1, q_2, q_3)$ in 3-way composition is augmented with a state $r$ of the filter automaton $W$, starting with state 0 of $W$. The transitions of the filter $W$ at each state $r$ determine the matches or moves allowed for that state $(q_1, q_2, q_3, r)$ of the composed machine.

## 5 Experiments

This section reports the results of experiments carried out in two different applications: the computation of a complex edit-distance between two automata, as motivated by applications in text and speech processing [9], and the computation of kernels between automata needed in spoken-dialog classification and other machine learning tasks.

### Table 1. Comparison of 3-way composition with standard composition. The computation times are reported in seconds, the size of $T_2$ in number of transitions. These experiments were performed on a dual-core AMD Opteron 2.2GHz with 16GB of memory, using the same software library and basic infrastructure.

| Size of $T_2$ | n-gram Kernel $\leq 2$ | n-gram Kernel $\leq 3$ | n-gram Kernel $\leq 4$ | n-gram Kernel $\leq 5$ | n-gram Kernel $\leq 6$ | n-gram Kernel $\leq 7$ | Edit distance standard | Edit distance +transpositions |
|---------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| Standard      | 65.3                    | 68.3                    | 71.0                    | 73.5                    | 76.3                    | 78.3                    | 586.1                   | 913.5                   |
| 3-way         | 8.0                     | 8.1                     | 8.2                     | 8.2                     | 8.2                     | 8.2                     | 3.8                     | 5.9                     |
| 70K           | 100K                    | 130K                    | 160K                    | 190K                    | 220K                    | 25M                     | 75M                     |

In the edit-distance case, the standard transducer $T_2$ used was one based on all insertions, deletions, and substitutions with different costs [9]. A more realistic transducer...
was one augmented with all transpositions, e.g., \(ab \rightarrow ba\), with different costs. In the kernel case, \(n\)-gram kernels with varying \(n\)-gram order were used [3].

Table 5 shows the results of these experiments. The finite automata \(T_1\) and \(T_3\) used were extracted from real text and speech processing tasks. The results show that in all cases, 3-way composition is orders of magnitude faster than standard composition.

6 Conclusion

We presented a general algorithm for the composition of weighted finite-state transducers. In many instances, 3-way composition benefits from a significantly better time and space complexity. Our experiments with both complex edit-distance computations arising in a number of applications in text and speech processing, and with kernel computations, crucial to many machine learning algorithms applied to sequence prediction, show that our algorithm is also substantially faster than standard composition in practice. We expect 3-way composition to further improve efficiency in a variety of other areas and applications in which weighted composition of transducers is used.

Acknowledgments. The research of Cyril Allauzen and Mehryar Mohri was partially supported by the New York State Office of Science Technology and Academic Research (NYS-TAR). This project was also sponsored in part by the Department of the Army Award Number W81XWH-04-1-0307. The U.S. Army Medical Research Acquisition Activity, 820 Chandler Street, Fort Detrick MD 21702-5014 is the awarding and administering acquisition office. The content of this material does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

1. Jean Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.
2. Stanley Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical Report, TR-10-98, Harvard University, 1998.
3. Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004.
4. Karel Culik II and Jarkko Kari. Digital Images and Formal Languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 599–616. Springer, 1997.
5. Samuel Eilenberg. *Automata, Languages and Machines*. Academic Press, 1974–76.
6. Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 35(3):400–401, 1987.
7. Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1986.
8. Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2), 1997.
9. Mehryar Mohri. Edit-Distance of Weighted Automata: General Definitions and Algorithms. *International Journal of Foundations of Computer Science*, 14(6):957–982, 2003.
10. Mehryar Mohri. Statistical Natural Language Processing. In M. Lothaire, editor, *Applied Combinatorics on Words*. Cambridge University Press, 2005.

11. 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)*. John Wiley and Sons, 1996.

12. Fernando Pereira and Michael Riley. *Finite State Language Processing*, chapter Speech Recognition by Composition of Weighted Finite Automata. The MIT Press, 1997.

13. Dominique Perrin. Words. In M. Lothaire, editor, *Combinatorics on words*, Cambridge Mathematical Library, Cambridge University Press, 1997.

14. Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, 1978.