Hyper-Minimization for Deterministic Weighted Tree Automata

Andreas Maletti∗
Universität Leipzig, Institute of Computer Science
Augustusplatz 10–11, 04109 Leipzig, Germany
maletti@informatik.uni-leipzig.de

Daniel Quernheim∗
Universität Stuttgart, Institute for Natural Language Processing
Pfaffenwaldring 5b, 70569 Stuttgart, Germany
daniel@ims.uni-stuttgart.de

Hyper-minimization is a state reduction technique that allows a finite change in the semantics. The
theory for hyper-minimization of deterministic weighted tree automata is provided. The presence of
weights slightly complicates the situation in comparison to the unweighted case. In addition, the first
hyper-minimization algorithm for deterministic weighted tree automata, weighted over commutative
semifields, is provided together with some implementation remarks that enable an efficient imple-
mentation. In fact, the same run-time \(O(m \log n)\) as in the unweighted case is obtained, where \(m\)
is the size of the deterministic weighted tree automaton and \(n\) is its number of states.

1 Introduction

Deterministic finite-state tree automata (DTA) \([13, 14]\) are one of the oldest, simplest, but most useful
devices in computer science representing structure. They have wide-spread applications in linguistic
analysis and parsing \([27]\) because they naturally can represent derivation trees of a context-free grammar.
Due to the size of the natural language lexicons and processes like state-splitting, we often obtain huge
DTA consisting of several million states. Fortunately, each DTA allows us to efficiently compute a unique
(up to isomorphism) equivalent minimal DTA, which is an operation that most tree automata toolkits
naturally implement. The asymptotically most efficient minimization algorithms are based on \([22, 18]\),
which in turn are based on the corresponding procedures for deterministic string automata \([20, 16, 30]\).
In general, all those procedures compute the equivalent states and merge them in time \(O(m \log n)\), where
\(n\) is the number of states of the input DTA and \(m\) is its size.

Hyper-minimization \([3]\) is a state reduction technique that can reduce beyond the classical minimal
device because it allows a finite change in the semantics (or a finite number of errors). It was already suc-
cessfully applied to a variety of devices such as deterministic finite-state automata \([12, 19]\), deterministic
tree automata \([21]\) as well as deterministic weighted automata \([24]\). With recent progress in the area of
minimization for weighted deterministic tree automata \([25]\), which provides the basis for this contribu-
tion, we revisit hyper-minimization for weighted deterministic tree automata. The asymptotically fastest
hyper-minimization algorithms \([12, 19]\) for DFA compute the “almost-equivalence” relation and merge
states with finite left language, called preamble states, according to it in time \(O(m \log n)\), where \(m\)
is the size of the input device and \(n\) is the number of its states. Naturally, this complexity is the goal for
our investigation as well. Variations such as cover automata minimization \([8]\), which has been explored
before hyper-minimization due to its usefulness in compressing finite languages, or \(k\)-minimization \([12]\)
restrict the length of the error strings instead of their number, but can also be achieved within the stated
time-bound.

As in \([24]\) our weight structures will be commutative semifields, which are commutative semi-
rings \([17, 15]\) with multiplicative inverses. As before, we will restrict our attention to deterministic

∗Both authors were financially supported by the German Research Foundation (DFG) grant MA / 4959 / 1–1.

This work is licensed under the Creative Commons Attribution License.
automata. Actually, the mentioned applications of DTA often use the weighted version to compute a quantitative answer (i.e., the numerically best-scoring parse, etc). We already know that weighted deterministic tree automata (DWTA) \cite{21,11} over semifields can be efficiently minimized \cite{25}, although the minimal equivalent DWTA is no longer unique due to the ability to “push” weights \cite{26,10,25}. The asymptotically fastest minimization algorithm \cite{25} nevertheless still runs in time $O(m \log n)$. To the authors’ knowledge, \cite{25} is currently the only published algorithm achieving this complexity for DWTA. Essentially, it normalizes the input DWTA by “pushing” weights, which yields that, in the process, the signatures of equivalent states become equivalent, so that a classical unweighted minimization can then perform the computation of the equivalence and the merges. To this end, it is important that the signature ignores states that can only recognize finitely many contexts, which are called co-preamble states, to avoid computing a wrong “pushing” weight.

We focus on an almost-equivalence notion that allows the recognized weighted tree languages to differ (in weight) for finitely many trees. Thus, we join the results on unweighted hyper-minimization for DTA \cite{21} and weighted hyper-minimization for WDFA \cite{24}. Our algorithms (see Algorithms 1 and 2) contain features of both of their predecessors and are asymptotically as efficient as them because they also run in time $O(m \log n)$. As in \cite{28}, albeit in a slightly different format, we use standardized signatures to avoid the explicit pushing of weights that was successful in \cite{25}. This adjustment allows us to mold our weighted hyper-minimization algorithm into the structure of the unweighted algorithm \cite{19}.

2 Preliminaries

We use $\mathbb{N}$ to denote the set of all nonnegative integers (including 0). For every integer $n \in \mathbb{N}$, we use the set $[n] = \{i \in \mathbb{N} | 1 \leq i \leq n\}$. Given two sets $S$ and $T$, their symmetric difference $S \triangle T$ is given by $S \triangle T = (S - T) \cup (T - S)$. An alphabet $\Sigma$ is simply a finite set of symbols, and a ranked alphabet $(\Sigma, \text{rk})$ consists of an alphabet $\Sigma$ and a ranking $\text{rk} : \Sigma \to \mathbb{N}$. We let $\Sigma_n = \{\sigma \in \Sigma | \text{rk}(\sigma) = n\}$ be the set of symbols of rank $n$ for every $n \in \mathbb{N}$. We often represent the ranked alphabet $(\Sigma, \text{rk})$ by $\Sigma$ alone and assume that the ranking ‘rk’ is implicit. Given a set $\mathcal{T}$ and a ranked alphabet $\Sigma$, we let

$$\Sigma(\mathcal{T}) = \{\sigma(t_1, \ldots, t_n) | n \in \mathbb{N}, \sigma \in \Sigma_n, t_1, \ldots, t_n \in \mathcal{T}\} \ .$$

The set $\mathcal{T}_\Sigma(Q)$ of $\Sigma$-trees indexed by a set $Q$ is the smallest set $T$ such that $Q \cup \Sigma(T) \subseteq T$. We write $\mathcal{T}_\Sigma$ for $\mathcal{T}_\Sigma(\emptyset)$. Given a tree $t \in \mathcal{T}_\Sigma(Q)$, its positions $\text{pos}(t) \subseteq \mathbb{N}^*$ are inductively defined by $\text{pos}(q) = \{\varepsilon\}$ for each $q \in Q$ and $\text{pos}(\sigma(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \{iw | i \in [n], w \in \text{pos}(t_i)\}$ for all $n \in \mathbb{N}$, $\sigma \in \Sigma_n$, and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(Q)$. For each position $w \in \text{pos}(t)$, we write $t(w)$ for the label of $t$ at position $w$ and $t|_w$ for the subtree of $t$ rooted at $w$. Formally,

$$q(\varepsilon) = q \quad \text{and} \quad (\sigma(t_1, \ldots, t_n))(w) = \begin{cases} \sigma & \text{if } w = \varepsilon \\ t_i(v) & \text{if } w = iv \text{ with } i \in [n], v \in \mathbb{N}^* \end{cases}$$

for all $q \in Q$, $n \in \mathbb{N}$, $\sigma \in \Sigma_n$, and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(Q)$. The height $\text{ht}(t)$ of a tree $t \in \mathcal{T}_\Sigma(Q)$ is simply $\text{ht}(t) = \max \{|w| | w \in \text{pos}(t)|\}$.

We reserve the use of the special symbol $\square$ of rank 0. A tree $t \in \mathcal{T}_{\Sigma,\{\square\}}(Q)$ is a $\Sigma$-context indexed by $Q$ if the symbol $\square$ occurs exactly once in $t$. The set of all $\Sigma$-contexts indexed by $Q$ is denoted by $C_\Sigma(Q)$. As
before, we write \( C_\Sigma \) for \( C_\Sigma(\emptyset) \). For each \( c \in C_\Sigma(Q) \) and \( t \in T_\Sigma(Q) \), the substitution \( c[t] \) denotes the tree obtained from \( c \) by replacing \( \emptyset \) by \( t \). Similarly, we use the substitution \( c[c'] \) with another context \( c' \in C_\Sigma(Q) \), in which case we obtain yet another context.

We take all weights from a **commutative semifield** \( \langle S, +, \cdot, 0, 1 \rangle \) \(^1\) which is an algebraic structure consisting of a commutative monoid \( (S, +, 0) \) and a commutative group \( (S - \{0\}, \cdot, 1) \) such that

- \( s \cdot 0 = 0 \) for all \( s \in S \), and
- \( s \cdot (s_1 + s_2) = (s \cdot s_1) + (s \cdot s_2) \) for all \( s, s_1, s_2 \in S \).

Roughly speaking, commutative semifields are commutative semirings \(^2\) with multiplicative inverses. Many practically relevant weight structures are commutative semifields. Examples include

- the real numbers \( \langle \mathbb{R}, +, \cdot, 0, 1 \rangle \),
- the tropical semifield \( \langle \mathbb{R} \cup \{\infty\}, \min, +, \infty, 0 \rangle \),
- the probabilistic semifield \( \langle [0,1], \max, \cdot, 0, 1 \rangle \) with \( [0,1] = \{ r \in \mathbb{R} | 0 \leq r \leq 1 \} \), and
- the **BOOLEAN semifield** \( \langle \{0,1\}, \max, \min, 0, 1 \rangle \).

For the rest of the paper, let \( \langle S, +, \cdot, 0, 1 \rangle \) be a commutative semifield (with \( 0 \neq 1 \)), and let \( _S = S - \{0\} \). For every \( s \in S \) we write \( s^{-1} \) for the inverse of \( s \); i.e., \( s \cdot s^{-1} = 1 \). For better readability, we will sometimes write \( \frac{s}{2} \) instead of \( s_1 \cdot s^{-1}_2 \). The following notions implicitly use the commutative semifield \( S \).

A weighted tree language is simply a mapping \( \varphi : T_\Sigma(Q) \to S \). Its **support** \( \text{supp}(\varphi) \subseteq T_\Sigma(Q) \) is \( \varphi^{-1}(S) \); i.e., the support contains exactly those trees that are evaluated to non-zero by \( \varphi \). Given \( s \in S \), we let \( (s \cdot \varphi): T_\Sigma(Q) \to S \) be the weighted tree language such that \( (s \cdot \varphi)(t) = s \cdot \varphi(t) \) for every \( t \in T_\Sigma(Q) \).

A deterministic weighted tree automaton (\( \text{DWT}A \)) \(^3\) \[^4\] \[^5\] \[^6\] \[^7\] \[^8\] is a tuple \( \mathcal{A} = (Q, \Sigma, \delta, \omega t, F) \) with

- a finite set \( Q \) of states,
- a ranked alphabet \( \Sigma \) of input symbols such that \( \Sigma \cap Q = \emptyset \),
- a transition mapping \( \delta : \Sigma(Q) \to Q \) \(^9\),
- a transition weight assignment \( \omega t : \Sigma(Q) \to _S \), and
- a set \( F \subseteq Q \) of final states.

The transition and transition weight mappings ‘\( \delta \)’ and ‘\( \omega t \)’ naturally extend to mappings \( \hat{\delta} : T_\Sigma(Q) \to Q \) and \( \hat{\omega t} : T_\Sigma(Q) \to S \) by

\[
\begin{align*}
\hat{\delta}(q) &= q \\
\hat{\delta}(\sigma(t_1, \ldots, t_n)) &= \delta(\hat{\delta}(t_1), \ldots, \hat{\delta}(t_n)) \\
\hat{\omega t}(q) &= 1 \\
\hat{\omega t}(\sigma(t_1, \ldots, t_n)) &= \omega t(\sigma(\hat{\delta}(t_1), \ldots, \hat{\delta}(t_n))) \cdot \prod_{i \in [n]} \hat{\omega t}(t_i)
\end{align*}
\]

for every \( q \in Q \), \( n \in \mathbb{N} \), \( \sigma \in \Sigma_n \), and \( t_1, \ldots, t_n \in T_\Sigma(Q) \). Since \( \hat{\delta}(t) = \delta(t) \) and \( \hat{\omega t}(t) = \omega t(t) \) for all \( t \in \Sigma(Q) \), we can safely omit the hat and simply write \( \delta \) and \( \omega t \) for \( \hat{\delta} \) and \( \hat{\omega t} \), respectively. The \( \text{DWT}A \) \( \mathcal{A} \) recognizes the weighted tree language \( \llbracket \mathcal{A} \rrbracket : T_\Sigma \to S \) such that

\[
\llbracket \mathcal{A} \rrbracket (t) = \begin{cases} 
\omega t(t) & \text{if } \delta(t) \in F \\
0 & \text{otherwise}
\end{cases}
\]

for all \( t \in T_\Sigma \). Two \( \text{DWT}A \) \( \mathcal{A} \) and \( \mathcal{B} \) are equivalent if \( \llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{B} \rrbracket \); i.e., their recognized weighted tree languages coincide. A \( \text{DWT}A \) over the **BOOLEAN** semifield \( \mathcal{B} \) is also called \( \text{DTA} \) \(^1\) \[^13\] \[^14\] and written \((Q, \Sigma, \delta, F)\) since the component ‘\( \omega t \)’ is uniquely determined. Moreover, we identify each **BOOLEAN**

\(^1\) We generally require \( 0 \neq 1 \), and in fact the additive monoid is rather irrelevant for our purposes.

\(^2\) Note that our \( \text{DWT}A \) are always total. We additionally disallow transition weight 0. If a transition is undesired, then its transition target can be set to a sink state, which we commonly denote by \( \perp \). Finally, the restriction to final states instead of final weights does not cause a difference in expressive power in our setting \[^6\] Lemma 6.1.4].
weighted tree language \( \varphi : T_\Sigma(Q) \to \{0,1\} \) with its support. Finally, the set \( C_\delta \) of shallow transition contexts is

\[ C_\delta = \{ \sigma(q_1, \ldots, q_{i-1}, q_i, q_{i+1}, \ldots, q_n) \mid n \in \mathbb{N}, i \in [n], \sigma \in \Sigma_n, q_1, \ldots, q_n \in Q \} , \]

which we assume to be totally ordered by some arbitrary order \( \leq \).

Intuitively, \( [q]_\varphi \) is the weighted (extended) language recognized by \( \varphi \) starting in state \( q \). Two states \( q, q' \in Q \) are equivalent \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \), written \( q \equiv q' \), if there exists \( s \in \mathbb{S} \) such that \( [q]_\varphi(c) = s \cdot [q']_\varphi(c) \) for all \( c \in C_\Sigma \). An equivalence relation \( \equiv \subseteq Q \times Q \) is a congruence relation (for the DWTAs \( \varphi \)) if for all \( n \in \mathbb{N} \), \( \sigma \in \Sigma_n \), and \( q_1 \equiv q_1', \ldots, q_n \equiv q_n' \) we have \( \delta(\sigma(q_1, \ldots, q_n)) \equiv \delta(\sigma(q_1', \ldots, q_n')) \). It is known \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \) that \( \equiv \) is a congruence relation. The DWTAs \( \varphi \) is minimal if there is no equivalent DWTAs with strictly fewer states. We can compute a minimal DWTAs efficiently using a variant of Hopcroft’s algorithm \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \) that computes \( \equiv \) and runs in time \( O(m \log n) \), where \( m = |\Sigma(Q)| \) is the size of \( \varphi \) and \( n = |Q| \).

3 A characterization of hyper-minimality

Hyper-minimization \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \) is a form of lossy compression that allows any finite number of errors. It has been investigated in \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \) for deterministic finite-state automata and in \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \) for deterministic tree automata. Finally, hyper-minimization was already generalized to weighted deterministic finite-state automata in \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \), from which we borrow much of the general approach. In the following, let \( \varphi = (Q, \Sigma, \delta, \text{wt}, F) \) and \( B = (P, \Sigma, \mu, \text{wt'}, \vartheta) \) be DWTAs over the commutative semifield \( (\mathbb{S}, +, \cdot, 0, 1) \) with \( 0 \neq 1 \).

We start with the basic definition of when two weighted tree languages are almost-equivalent. We decided to use the same approach as in \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \), so we require that the weighted tree languages, seen as functions, must coincide on almost all trees. Note that this restriction is not simply the same as requiring that the weighted tree languages have almost-equal (i.e., finite-difference) supports. It fact, our definition yields that the supports are almost-equal, but that is not sufficient. In addition, we immediately allow a scaling factor in many of our basic definitions since those are already required in classical minimization \( \footnote{\text{Almost all" means all but a finite number, as usual.}} \) to obtain the most general statements. Naturally, a scaling factor is not allowed for the almost-equivalence of DWTAs since these are indeed supposed assign a different weight to only finitely many trees.

**Definition 1.** Two weighted tree languages \( \varphi_1, \varphi_2 : T_\Sigma(Q) \to \mathbb{S} \) are almost-equivalent, written \( \varphi_1 \approx \varphi_2 \), if there exists \( s \in \mathbb{S} \) such that \( \varphi_1(t) = s \cdot \varphi_2(t) \) for almost all \( t \in T_\Sigma(Q) \). We write \( \varphi_1 \approx_\varphi \varphi_2 \) (\( s \)) to indicate the factor \( s \). The DWTAs \( \varphi \) and \( B \) are almost-equivalent if \( [\varphi] \approx [B] \) (1). Finally, the states \( q \in Q \) and \( p \in P \) are almost-equivalent if there exists \( s \in \mathbb{S} \) such that \( [q]_\varphi(c) = s \cdot [p]_\varphi(c) \) for almost all \( c \in C_\Sigma \).

We start with some basic properties of \( \approx \), which is shown to be an equivalence relation both on the weighted tree languages as well as on the states of a single DWTAs. In addition, we demonstrate that the latter version is even a congruence relation. This shows that once we are in almost-equivalent states, the same impetus causes the different devices to switch to other almost-equivalent states.
Lemma 2. Almost-equivalence is an equivalence relation such that $\delta(c[q]) \approx \mu(c[p])$ for all $c \in C_S$, $q \in Q$, and $p \in P$ with $q \approx p$.

Proof. Trivially, $\approx$ is reflexive and symmetric (because we have multiplicative inverses for all elements of $S$). Let $\varphi_1, \varphi_2, \varphi_3 : T_S(Q) \to S$ be weighted tree languages such that $\varphi_1 \approx \varphi_2$ (s) and $\varphi_2 \approx \varphi_3$ (s') for some $s, s' \in S$. Then there exist finite sets $L, L' \subseteq T_S(Q)$ such that $\varphi_1(t) = s \cdot \varphi_2(t)$ and $\varphi_2(t') = s' \cdot \varphi_3(t')$ for all $t \in T_S(Q) - L$ and $t' \in T_S(Q) - L'$. Consequently, $\varphi_1(t'') = s \cdot s' \cdot \varphi_3(t'')$ for all $t'' \in T_S(Q) - (L \cup L')$, which proves $\varphi_1 \approx \varphi_3$ (s $\cdot$ s') and thus transitivity. Hence, $\approx$ is an equivalence relation. The same arguments can be used for $\approx$ on DWTA and states. For the second property, induction allows us to easily prove \[ (6) \] that

$$ [q]_{\mathcal{A}}(c'[c]) = \text{wt}(c[q]) \cdot [\delta(c[q])]_{\mathcal{A}}(c') \quad \text{and} \quad [p]_{\mathcal{A}}(c_2[c_1]) = \text{wt}(c_1[p]) \cdot [\mu(c_1[p])]_{\mathcal{A}}(c_2) \quad (†) $$

for all $c, c' \in C_S(Q)$ and $c_1, c_2 \in C_S(P)$. Since $q \approx p$ (s), there exists a finite set $C \subseteq C_S$ such that $[q]_{\mathcal{A}}(c'') = s \cdot [p]_{\mathcal{A}}(c'')$ for all $c'' \in C_S - C$. Consequently,

$$ [\delta(c[q])]_{\mathcal{A}}(c') = [q]_{\mathcal{A}}(c'[c]) = s \cdot [p]_{\mathcal{A}}(c'[c]) = s \cdot \frac{\text{wt}(c[p])}{\text{wt}(c[q])} \cdot [\mu(c[p])]_{\mathcal{A}}(c') $$

for all $c' \in C_S$ such that $c'[c] \notin C$, which proves that $\delta(c[q]) \approx \mu(c[p])$. \[ \square \]

Next, we show that almost-equivalent states of the same DWTA even coincide (up to the factor $s$) on almost all extended contexts, which are contexts in which states may occur.

Lemma 3. Let $\mathcal{A}$ be minimal and $q \approx q'(s)$ for some $s \in S$ and $q, q' \in Q$. Then

$$ [q]_{\mathcal{A}}(c) = s \cdot [q']_{\mathcal{A}}(c) \quad (‡) $$

for almost all $c \in C_S(Q)$.

Proof. By definition of $q \approx q'(s)$, there exists a finite set $C \subseteq C_S$ such that $[q]_{\mathcal{A}}(c) = s \cdot [q']_{\mathcal{A}}(c)$ for all $c \in C_S - C$. Let $h \geq \max \{ \text{ht}(c) \mid c \in C \}$ be an upper bound for the height of those finitely many contexts. Clearly, there are only finitely many contexts of $C_S$ that have height at most $h$. Now, let $c \in C_S(Q)$ be an extended context such that $\text{ht}(c) > h$, and let $W = \{ w \in \text{pos}(c) \mid c(w) \in Q \}$ be the positions that are labeled with states. For each state $q \in Q$, select $t_q \in \delta^{-1}(q) \cap T_S$ a tree (without occurrences of states) that is processed in $q$. Clearly, such a tree exists for each state because $\mathcal{A}$ is minimal. Let $c'$ be the context obtained from $c$ by replacing each state occurrence of $q$ by $t_q$. Obviously, $\text{ht}(c') \geq \text{ht}(c) > h$ because we replace only leaves. Consequently, $c' \in C_S - C$. Using a variant \[ (6) \] of \[ (†) \] we obtain

$$ [q]_{\mathcal{A}}(c) \cdot \prod_{w \in W} \text{wt}(t_{c(w)}) = [q]_{\mathcal{A}}(c') = s \cdot [q']_{\mathcal{A}}(c') = s \cdot [q']_{\mathcal{A}}(c) \cdot \prod_{w \in W} \text{wt}(t_{c(w)}) \ , $$

where the second equality is due to the fact that $c' \in C_S - C$. Comparing the left-hand and right-hand side and cancelling the additional terms, which is allowed in a commutative semifield, we obtain $[q]_{\mathcal{A}}(c) = s \cdot [q']_{\mathcal{A}}(c)$ for all $c \in C_S(Q)$ with $\text{ht}(c) > h$, and thus for almost all $c \in C_S(Q)$ as required. \[ \square \]

\*\*Note that $1^{-1} = 1$ and $1 \cdot 1 = 1$, so the restriction to factor 1 in the definition of the almost-equivalence of DWTA is not problematic.\*\*
As in all the other scenarios, the goal of hyper-minimization given device $\mathcal{A}$ is to construct an almost-equivalent device $\mathcal{B}$ such that no device is smaller than $\mathcal{B}$ and almost-equivalent to $\mathcal{A}$. In our setting, the devices are DWTA over the ranked alphabet $\Sigma$ and the commutative semifield $\mathbb{S}$. Since almost-equivalence is an equivalence relation by Lemma 2, we can replace the requirement “almost-equivalent to $\mathcal{A}$” by “almost-equivalent to $\mathcal{B}$” and call a DWTA $\mathcal{B}$ hyper-minimal if no (strictly) smaller DWTA is almost-equivalent to it. Then hyper-minimization equates to the computation of a hyper-minimal DWTA $\mathcal{B}$ that is almost-equivalent to $\mathcal{A}$. Let us first investigate hyper-minimality, which was characterized in [3] for the BOOLEAN semifield using the additional notion of a preamble state.

**Lemma 6.** Let $q \in Q$ be such that $q \rightarrow$ be such that $q \rightarrow$ weighted merges. factors, so we use the weighted merges just introduced. The next lemma hints at the correct use of

In other words, a state is a preamble state if and only if it accepts finitely many trees (without occurrences of states). This notion is essentially unweighted, so the discussion in [21] applies. In particular, we can compute the set of kernel states in time $O(m)$ with $m = |\Sigma(Q)|$ being the size of the DWTA $\mathcal{A}$.

Recall that a DWTA (without unreachable states; i.e., $\delta^{-1}(q) \cap T_{\Sigma} \neq \emptyset$ for every $q \in Q$) is minimal if and only if it does not have a pair of different, but equivalent states [7,5]. The “only-if” part of this statement is shown by merging two equivalent states to obtain a smaller, but equivalent DWTA. Let us define a merge that additionally applies a weight $s$ to the rerouted transitions.

**Definition 4** (see [3] Definition 2.11). A state $q \in Q$ is a preamble state if $\delta^{-1}(q) \cap T_{\Sigma}$ is finite. Otherwise, it is a kernel state.

In our approach to weighted hyper-minimization, we also merge, but we need to take care of the factors, so we use the weighted merges just introduced. The next lemma hints at the correct use of weighted merges.

**Lemma 5.** Let $q, q' \in Q$ and $s \in \mathbb{S}$ with $q \neq q'$. The $s$-weighted merge of $q$ into $q'$ is the DWTA merge $\delta' \in Q_{Q} = \{Q - \{q\}, \Sigma, \delta', w', F - \{q\}\}$ such that for all $t \in \Sigma(Q - \{q\})$

$$
\delta'(t) = \begin{cases} q' & \text{if } \delta(t) = q \\ \delta(t) & \text{otherwise} \end{cases}
$$

$$
w'(t) = \begin{cases} s \cdot w(t) & \text{if } \delta(t) = q \\ w(t) & \text{otherwise.} \end{cases}
$$

In our approach to weighted hyper-minimization, we also merge, but we need to take care of the factors, so we use the weighted merges just introduced. The next lemma hints at the correct use of weighted merges.

**Lemma 6.** Let $q, q' \in Q$ be different states, of which $q$ is a preamble state, and $s \in \mathbb{S}$ be such that $q \approx q'(s)$. Then merge $\delta'(q \rightarrow q')$ is almost-equivalent to $\mathcal{A}$.

**Proof.** Since $q \approx q'(s)$, there exists a finite set $C \subseteq C_{\Sigma}$ such that $[q]_{\mathcal{A}}(c) = s \cdot [q]_{\mathcal{A}}(c)$ for all $c \in C_{\Sigma} - C$. Let $h \geq \max \{ \text{ht}(c) \mid c \in C \}$ be an upper bound on the height of the contexts of $C$. Moreover, let $h' \geq \max \{ \text{ht}(t) \mid t \in \delta^{-1}(q) \cap T_{\Sigma} \}$ be an upper bound for the height of the trees of $\delta^{-1}(q) \cap T_{\Sigma}$, which is a finite set since $q$ is a preamble state. Finally, let $z > h + h'$. Now we return to the main claim. Let $\mathcal{B} = \text{merge}_{\mathcal{A}}(q \rightarrow q')$ and consider an arbitrary tree $t \in T_{\Sigma}$ whose height is at least $z$. Clearly, showing that $\mathcal{B}(t) = \mathcal{A}(t)$ for all trees $t$ with $\text{ht}(t) \geq z$ proves that $\mathcal{B}$ and $\mathcal{A}$ are almost-equivalent.

Let $W = \{ w \in \text{pos}(t) \mid \delta(t|_{w}) = q \}$ be the set of positions of the subtrees that are recognized in state $q$. Now $\text{wt}'(t|_{w}) = s \cdot \text{wt}(t|_{w})$ for all $w \in W$ because clearly the subtrees $t|_{w}$ only use states different from $q$ except at the root, where $\mathcal{A}$ switches to $q$ and $\mathcal{B}$ switches to $q'$ with the additional weight $s$. Note that $q$ cannot occur anywhere else inside those subtrees because this would create a loop which is impossible for a preamble state. Let $W = \{ w_1, \ldots, w_m \}$ with $w_1 \subseteq \cdots \subseteq w_m$, in which $\subseteq$ is the lexicographic order on $\mathbb{N}^{*}$. Let $c_1 \in C_{\Sigma}$ be the context obtained by removing the subtree at $w_1$ from $t$. Note that $c_1$ is taller.

---

5There are only finitely many ranked trees up to a certain height and recall that almost-equivalence does not permit a scaling factor for DWTA.
than \( h \) (i.e., \( \text{ht}(c_1) > h \)) and thus \( c_1 \in C_\Sigma - C \) because the height of \( t \) is larger than \( h + h' \) and the height of \( t|_{w_1} \) is at most \( h' \). Consequently, using a variant of (5) we obtain

\[
A(t) = A(c_1|_{t|_{w_1}}) = \text{wt}(t|_{w_1}) \cdot [q]_A(c_1) = \frac{\text{wt}'(t|_{w_1})}{s} \cdot [q']_A(c_1) = \text{wt}'(t|_{w_1}) \cdot [q']_A(c_1)
\]

= \text{wt}'(t|_{w_1}) \cdot \begin{cases} \text{wt}(c_1[q']) & \text{if } \delta(c_1[q']) \in F \\ 0 & \text{otherwise}. \end{cases}

Let \( c_2 \) be the context obtained from \( c_1[q'] \) by replacing the subtree at \( w_2 \) by \( \Box \). Also \( c_2 \notin C \).

\[
= \text{wt}'(t|_{w_1}) \cdot \begin{cases} \text{wt}(c_2[t_{w_2}]) & \text{if } \delta(c_2[t_{w_2}]) \in F \\ 0 & \text{otherwise.} \end{cases} = \text{wt}'(t|_{w_1}) \cdot \text{wt}(t|_{w_2}) \cdot [q]_A(c_2)
\]

\[
\doteq \frac{\text{wt}'(t|_{w_1})}{s} \cdot [q']_A(c_2) = \text{wt}'(t|_{w_1}) \cdot \text{wt}(t|_{w_2}) \cdot [q']_A(c_2),
\]

which can now be iterated to obtain

\[
= \text{wt}'(t|_{w_1}) \cdot \ldots \cdot \text{wt}'(t|_{w_m}) \cdot [q']_A(c_m) = \text{wt}'(t|_{w_1}) \cdot \ldots \cdot \text{wt}'(t|_{w_m}) \cdot [q']_A(c_m) = A(t),
\]

where the second-to-last step is justified because the state \( q \) is not used when processing the context \( c_m \). This proves the statement.

**Theorem 7.** A minimal DWTA is hyper-minimal if and only if it has no pair of different, but almost-equivalent states, of which at least one is a preamble state.

**Proof.** Let \( A \) be the minimal DWTA. For the “only if” part, we know by Lemma\(^6\) that the smaller DWTA \( \merge_A(q \rightarrow q') \) is almost-equivalent to \( A \) if \( q \approx q' \) (s) and \( q \) is a preamble state. For the “if” direction, suppose that \( B \) is almost-equivalent to \( A \) and \( |P| < |Q|^2 \). For all \( t \in T_\Sigma \) we have \( \delta(t) \approx \mu(t) \) by Lemma\(^2\).

Since \( |P| < |Q| \), there exist \( t_1, t_2 \in T_\Sigma \) with \( q_1 = \delta(t_1) \neq \delta(t_2) = q_2 \) but \( \mu(t_1) = \mu(t_2) \). Consequently, \( q_1 = \delta(t_1) \approx \mu(t_1) = \mu(t_2) \approx \delta(t_2) = q_2 \), which yields \( q_1 \approx q_2 \). By assumption, \( q_1 \) and \( q_2 \) are kernel states. Using a variation of the above argument (see \(^3\) Theorem 3.3) we can obtain \( t_1 \) and \( t_2 \) with the above properties such that \( \text{ht}(t_1), \text{ht}(t_2) \geq |Q|^2 \). Due to their heights, we can pump the trees \( t_1 \) and \( t_2 \), which yields that the states \( \langle q_1, p \rangle \) and \( \langle q_2, p \rangle \) are kernel states of the HADAMARD product \( A \cdot B \). Since \( A \) and \( B \) are almost-equivalent, we have

\[
\text{wt}(t_1) \cdot [q_1]_A(c) = \text{wt}(t_2) \cdot [q_2]_A(c)
\]

for almost all \( c \in C_\Sigma \) using again the tree variant of (5). Moreover, since both \( \langle q_1, p \rangle \) and \( \langle q_2, p \rangle \) are kernel states, we can select \( t_1 \) and \( t_2 \) such that the previous statements are actually true for all \( c \in C_\Sigma \). Consequently,

\[
\frac{\text{wt}(t_1)}{\text{wt}(t_1)} \cdot [q_1]_A(c) = \frac{\text{wt}(t_2)}{\text{wt}(t_2)} \cdot [q_2]_A(c)
\]

for all \( c \in C_\Sigma \) and \( s = \frac{\text{wt}(t_1)}{\text{wt}(t_1)} \cdot \frac{\text{wt}(t_2)}{\text{wt}(t_2)} \), which yields \( q_1 \approx q_2 \). This contradicts minimality since \( q_1 \approx q_2 \), which shows that such a DWTA \( B \) cannot exist.

\(^6\)Recall that almost-equivalent DWTA do not permit a scaling factor; their semantics need to coincide for almost all trees.
Algorithm 1 Structure of the hyper-minimization algorithm.

Require: a DWTA $\mathcal{A}$ with $n$ states

Return: an almost-equivalent hyper-minimal DWTA $\mathcal{A}$

$\mathcal{A} \leftarrow \text{MINIMIZE}(\mathcal{A})$ \hspace{1cm} // $O(m \log n)$

2: $K \leftarrow \text{COMPUTEKERNEL}(\mathcal{A})$ \hspace{1cm} // $O(m)$

3: $K \leftarrow \text{COMPUTEKORENL}(\mathcal{A})$ \hspace{1cm} // $O(m)$

4: $(\sim, t) \leftarrow \text{COMPUTEAALMOSTEQUIVALENCE}(\mathcal{A}, K)$ \hspace{1cm} // Algorithm 2 — $O(m \log n)$

return $\text{MERGESTATES}(\mathcal{A}, K, \sim, t)$ \hspace{1cm} // Algorithm 3 — $O(m)$

4 Hyper-minimization

Next, we consider some algorithmic aspects of hyper-minimization for DWTA. Since the unweighted case is already well-described in the literature [21], we focus on the weighted case, for which we need the additional notion of co-preamble states [24], which in analogy to [24] are those states with finite support of their weighted context language. Let $P$ and $K$ be the sets of preamble and kernel states of $\mathcal{A}$, respectively.

Definition 8. A state $q \in Q$ is a co-preamble state if $\text{supp}([q]_\mathcal{A})$ is finite. Otherwise it is a co-kernel state. The sets of all co-preamble states and all co-kernel states are $P$ and $K = Q - P$, respectively.

Transitions entering a co-preamble state can be ignored while checking almost-equivalence because (up to a finite number of weight differences) the reached states behave like the sink state $\bot$. Trivially, all co-preamble states are almost-equivalent. In addition, a co-preamble state cannot be almost-equivalent to a co-kernel state. The interesting part of the almost-equivalence is thus completely determined by the weighted languages of the co-kernel states. This special role of the co-preamble states has already been pointed out in [12] in the context of DFA.

All hyper-minimization algorithms [3, 2, 12, 19] share the same overall structure (Algorithm 1). In the final step we perform state merges (see Definition 5). Merging only preamble states into almost-equivalent states makes sure that the resulting DWTA is almost-equivalent to the input DWTA by Lemma 6. Algorithm 1 first minimizes the input DWTA using, for example, the algorithm of [25]. With the help of a weight redistribution along the transitions (pushing), it reduces the problem to DTA minimization, for which we can use a variant of Hopcroft’s algorithm [18]. In the next step, we compute the set $K$ of kernel states of $\mathcal{A}$ [21] using any algorithm that computes strongly connected components (for example, Tarjan’s algorithm [29]). By [21] a state is a kernel state if and only if it is reachable from (i) a nontrivial strongly connected component or (ii) a state with a self-loop. Essentially, the same approach can be used to compute the co-kernel states. In line 4 we compute the almost-equivalence on the states $Q$, which is the part where the algorithms [3, 2, 12, 19] differ. Finally, we merge almost-equivalent states according to Lemma 6 until the obtained DWTA is hyper-minimal (see Theorem 7).

Lemma 9. Let $\mathcal{A}$ be a minimal DWTA. The states $q, q' \in Q$ are almost-equivalent if and only if there is $n \in \mathbb{N}$ such that $\delta(c[q]) = \delta(c[q'])$ for all $c \in C_\Sigma$ such that $\Box$ occurs at position $w$ in $c$ with $|w| \geq n$.

Our algorithm for computing the almost-equivalence is an extension of the algorithm of [24]. As in [24], we need to handle the scaling factors, for which we introduced the standardized signature in [24]. Roughly speaking, we ignore transitions into co-preamble states and normalize the transition weights. Recall that $C_\delta$ is the set of transition contexts; i.e., transitions with exactly one occurrence of the symbol $\Box$. Moreover, for every $q \in Q$, we let $c_q$ be the smallest transition context $c_q \in C_\delta$ such that
\[ \delta(c_0[q]) \in \mathcal{K}, \text{ where the total order on } C_\delta \text{ is arbitrary as assumed earlier, but it needs to be consistently used.} \]

**Definition 10.** Given \( q \in Q \), its standardized signature is
\[
\text{Sig}(q) = \left\{ \langle c, \delta(c[q]), \frac{\text{wt}(c[q])}{\text{wt}(c_0[q])} \rangle \mid c \in C_\delta, \delta(c[q]) \in \mathcal{K} \right\}.
\]

Next, we show that states with equal standardized signature are indeed almost-equivalent.

**Lemma 11.** For all \( q, q' \in Q \), if \( \text{Sig}(q) = \text{Sig}(q') \), then \( q \approx q' \).

**Proof.** If \( q \) or \( q' \) is a co-preamble state, then both \( q \) and \( q' \) are co-preamble states and thus \( q \approx q' \). Now, let \( q, q' \in \mathcal{K} \), and let \( c_q \in C_\delta \) be the smallest transition context such that \( c_q[q] \in \mathcal{K} \). Since \( q' \) has the same signature, \( c_q = c_{q'} \). In addition, let \( s = \frac{\text{wt}(c_q[q])}{\text{wt}(c_q[q'])} \). For every \( c \in C_\delta \) and \( c' \in C_\Sigma \),
\[
[q]_{\mathcal{A}}(c'[c]) \overset{\dagger}{=} \frac{\text{wt}(c[q])}{\text{wt}(c_0[q])} \cdot \text{wt}(c_0[q]) \cdot \frac{\text{wt}(c_0[q'])}{\text{wt}(c_q[q'])} = s \cdot \text{wt}(c_0[q]) \cdot [q']_{\mathcal{A}}(c')
\]

First, let \( \langle c, q_c, s_c \rangle \notin \text{Sig}(q) = \text{Sig}(q') \) for all \( q_c \in Q \) and \( s_c \in S \). Then \( c \) takes both \( q \) and \( q' \) into a co-preamble state and thus \( \langle q]_{\mathcal{A}}(c'[c]) = 0 = s \cdot [q']_{\mathcal{A}}(c'[c]) \) for almost all \( c' \in C_\Sigma \). Second, suppose that \( \langle c, q_c, s_c \rangle \in \text{Sig}(q) = \text{Sig}(q') \) for some \( q_c \in Q \) and \( s_c \in S \). Since \( \delta(c[q]) = q_c = \delta(c[q']) \), and we obtain
\[
[q]_{\mathcal{A}}(c'[c]) = \frac{\text{wt}(c[q])}{\text{wt}(c_0[q])} \cdot \text{wt}(c_0[q]) \cdot \frac{\text{wt}(c_0[q'])}{\text{wt}(c_q[q'])} = s \cdot \text{wt}(c_0[q]) \cdot [q']_{\mathcal{A}}(c') = s \cdot [q']_{\mathcal{A}}(c'[c])
\]

for every \( c' \in C_\Sigma \), which shows that \( q \approx q' \) (s) because the scaling factor \( s \) does not depend on the transition context \( c \).

In fact, the previous proof can also be used to show that at most the empty context \( \square \) yields a difference in the weighted context languages \( [q]_{\mathcal{A}} \) and \( [q']_{\mathcal{A}} \) (up to the common factor). For the completeness, we also need a (restricted) converse for minimal DWTA, which shows that as long as there are almost-equivalent states, we can also identify them using the standardized signature.

**Lemma 12.** Let \( \mathcal{A} \) be minimal, and let \( q \approx q' \) be such that \( \text{Sig}(q) \neq \text{Sig}(q') \). Then there exist \( r, r' \in Q \) such that \( r \neq r' \) and \( \text{Sig}(r) = \text{Sig}(r') \).

**Proof.** Since \( q \approx q' \), there exists an integer \( h \) such that \( \delta(c[q]) = \delta(c[q']) \) for all \( c \in C_\Sigma \) such that \( w \in \text{pos}(c) \) with \( c(w) = \square \) and \( |w| \geq h \) by Lemma 9. Let \( c' \in C_\Sigma \) be a maximal context such that \( r = \delta(c'[q]) \neq \delta(c'[q']) = r' \). Since \( c' \) is maximal, we have \( \delta(c''[c'[q]]) = q_{c''} = \delta(c''[c'[q']]) \) for all \( c'' \in C_\delta \). If \( q_{c''} \) is a co-preamble state, then \( \langle c, q_{c''}, s_c \rangle \notin \text{Sig}(r) = \text{Sig}(r') \) for all \( q_c \in Q \) and \( s_c \in S \). On the other hand, let \( q_{c''} \) be a co-kernel state, and let \( c_r \in C_\delta \) be the smallest transition context such that \( \delta(c_r[r]) \in \mathcal{K} \). Since \( q \approx q' \) and \( \approx \) is a congruence relation by Lemma 2, we have \( r \approx r' \) (s) for some \( s \in S \), which means that \( [r]_{\mathcal{A}}(c) = s \cdot [r']_{\mathcal{A}}(c) \) for almost all \( c \in C_\Sigma \). Consequently,
\[
\text{wt}(c''[r]) \cdot [q_{c''}]_{\mathcal{A}}(c) = s \cdot \text{wt}(c''[r']) \cdot [q_{c''}]_{\mathcal{A}}(c)
\]

\[
\text{wt}(c_r[r]) \cdot [\delta(c_r[r])]_{\mathcal{A}}(c) = s \cdot \text{wt}(c_r[r']) \cdot [\delta(c_r[r])]_{\mathcal{A}}(c)
\]
Algorithm 2 Algorithm computing the almost-equivalence \( \approx \) and scaling map \( f \).

**Require:** minimal DWTA \( A \) and its co-kernel states \( \overline{K} \)

**Return:** almost-equivalence \( \approx \) as a partition and scaling map \( f : Q \to \overline{K} \)

```plaintext
for all \( q \in Q \) do
  2: \( \pi(q) \leftarrow \{ q \}; \quad f(q) \leftarrow 1 \) // trivial initial blocks
  \( h \leftarrow 0; \quad I \leftarrow Q \) // hash map of type \( h : \text{Sig} \to Q \)
for all \( q \in I \) do
  4: \( \text{succ} \leftarrow \text{Sig}(q) \) // compute standardized signature using current \( \delta \) and \( \overline{K} \)
  6: if \( \text{HASVALUE}(h, \text{succ}) \) then
    \( q' \leftarrow \text{GET}(h, \text{succ}) \) // retrieve state in bucket ‘succ’ of \( h \)
    8: if \( |\pi(q')| \geq |\pi(q)| \) then
      \( \text{SWAP}(q, q') \) // exchange roles of \( q \) and \( q' \)
    10: \( I \leftarrow I \cup \{ r \in Q - \{ q' \} \mid \exists c \in C_{\delta} : \delta(c[r]) = q' \} \) // add predecessors of \( q' \)
    \( f(q') \leftarrow \frac{\text{wt}(c[q])}{\text{wt}(c[q'])} f(q'), \) // \( c_q \) is as in Definition 10
  12: \( A \leftarrow \text{merge}(_A f(q'), q) \) // merge \( q' \) into \( q \)
  14: \( \pi(q) \leftarrow \pi(q) \cup \pi(q') \) // \( q \) and \( q' \) are almost-equivalent
    for all \( r \in \pi(q') \) do
      \( f(r) \leftarrow f(r) \cdot f(q') \) // recompute scaling factors
  16: \( h \leftarrow \text{PUT}(h, \text{succ}, q) \) // store \( q \) in \( h \) under key ‘succ’
return \( (\pi, f) \)
```

for almost all \( c \in C_{\delta} \). Since both \( q_{\varepsilon^r} \) and \( \delta(c_r[r]) \) are co-kernel states, we immediately can conclude that
\[
\text{wt}(c''[r]) = s \cdot \text{wt}(c''[r']) \quad \text{and} \quad \text{wt}(c_r[r]) = s \cdot \text{wt}(c_r[r'])
\]
which yields
\[
\frac{\text{wt}(c''[r])}{\text{wt}(c_r[r])} = \frac{s \cdot \text{wt}(c''[r'])}{s \cdot \text{wt}(c_r[r'])} = \frac{\text{wt}(c''[r'])}{\text{wt}(c_r[r'])}.
\]

This proves \( \text{Sig}(r) = \text{Sig}(r') \) as required. \( \square \)

Lemmata [11] and [12] suggest Algorithm 2 for computing the almost-equivalence and a map representing the scaling factors. This map contains a scaling factor for each state with respect to a representative state of its block. Algorithm 2 is a straightforward modification of an algorithm by [19] using our standardized signatures. We first compute the standardized signature for each state and store it into a (perfect) hash map [9] to avoid pairwise comparisons. If we find a collision (i.e., a pair of states with the same signature), then we merge them such that the state representing the bigger block survives (see Lines 9 and 12). Each state is considered at most \( \log n \) times because the size of the “losing” block containing it at least doubles. After each merge, scaling factors of the “losing” block are computed with respect to the new representative. Again, we only recompute the scaling factor of each state at most \( \log n \) times. Hence the small modifications compared to [19] do not increase the asymptotic run-time of Algorithm 2 which is \( \mathcal{O}(n \log n) \) where \( n \) is the number of states (see Theorem 9 in [19]). Alternatively, we can use the standard reduction to a weighted finite-state automaton using each transition context \( c \in C_{\delta} \) as a new symbol.
Theorem 16. Algorithm $\text{Algorithm 2}$ computes $\approx$ and a scaling map.

Proof sketch. If there exist different, but almost-equivalent states, then there exist different states with the same standardized signature by Lemma 12. Lemma 11 shows that such states are almost-equivalent.
Finally, Lemma 15 shows that we can continue the computation of the almost-equivalence after a weighted merge of such states. The correctness of the scaling map is shown implicitly in the proof of Lemma 11.

**Theorem 17.** We can hyper-minimize $\text{DWTA}$ in time $O(m \log n)$, where $m = |\Sigma(Q)|$ and $n = |Q|$.

**References**


Hyper-minimization for deterministic weighted tree automata


