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 implementation. In fact, the same run-time $\mathcal{O}(m \log n)$ 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 $\mathcal{O}(m \log n)$, 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 successfully 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 contribution, 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 $\mathcal{O}(m \log n)$, 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 semirings [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.

Z. Ésik and Z. Fülöp (Eds.): Automata and Formal Languages 2014 (AFL 2014).
EPTCS 151, pp. 314–326, 2014. doi: 10.4204/EPTCS.151.22

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) [4, 11] over semifields can be efficiently minimized [25], although the minimal equivalent DWTA is no longer unique due to the ability to "push" weights [26, 10, 25]. The asymptotically fastest minimization algorithm [25] nevertheless still runs in time $\mathcal{O}(m \log n)$. To the authors' knowledge, [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 [21] and weighted hyper-minimization for WDFA [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 $\mathcal{O}(m \log n)$. As in [28], albeit in a slightly different format, we use standardized signatures to avoid the explicit pushing of weights that was successful in [25]. This adjustment allows us to mold our weighted hyper-minimization algorithm into the structure of the unweighted algorithm [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 $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Given two sets S and T, their *symmetric difference* $S \ominus T$ is given by $S \ominus T = (S - T) \cup (T - S)$. An alphabet Σ is simply a finite set of symbols, and a *ranked alphabet* (Σ, rk) consists of an alphabet Σ and a ranking $\mathrm{rk}\colon \Sigma \to \mathbb{N}$. We let $\Sigma_n = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = n\}$ be the set of symbols of rank n for every $n \in \mathbb{N}$. We often represent the ranked alphabet (Σ, rk) by Σ alone and assume that the ranking 'rk' is implicit. Given a set T and a ranked alphabet Σ, we let

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

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

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

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

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

We reserve the use of the special symbol $\square$ of rank 0. A tree $t \in T_{\Sigma \cup \{\square\}}(Q)$ is a *Σ-context indexed by Q* if the symbol $\square$ occurs exactly once in t. The set of all Σ-contexts indexed by Q is denoted by $C_\Sigma(Q)$. As before, we write C_Σ 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 $\square$ 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* $\langle \mathbb{S}, +, \cdot, 0, 1\rangle$,[1] which is an algebraic structure consisting of a commutative monoid $\langle \mathbb{S}, +, 0\rangle$ and a commutative group $\langle \mathbb{S} - \{0\}, \cdot, 1\rangle$ such that

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

Roughly speaking, commutative semifields are commutative semirings [17, 15] 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 $[0,1] = \{r \in \mathbb{R} \mid 0 \leq r \leq 1\}$, and
- the BOOLEAN semifield $\mathbb{B} = \langle \{0,1\}, \max, \min, 0, 1\rangle$.

For the rest of the paper, let $\langle \mathbb{S}, +, \cdot, 0, 1\rangle$ be a commutative semifield (with $0 \neq 1$), and let $\underline{\mathbb{S}} = \mathbb{S} - \{0\}$. For every $s \in \underline{\mathbb{S}}$ we write s^{-1} for the inverse of s; i.e., $s \cdot s^{-1} = 1$. For better readability, we will sometimes write $\frac{s_1}{s_2}$ instead of $s_1 \cdot s_2^{-1}$. The following notions implicitly use the commutative semifield $\mathbb{S}$. A *weighted tree language* is simply a mapping $\varphi \colon T_\Sigma(Q) \to \mathbb{S}$. Its *support* $\mathrm{supp}(\varphi) \subseteq T_\Sigma(Q)$ is $\mathrm{supp}(\varphi) = \varphi^{-1}(\underline{\mathbb{S}})$; i.e., the support contains exactly those trees that are evaluated to non-zero by φ. Given $s \in \mathbb{S}$, we let $(s \cdot \varphi) \colon T_\Sigma(Q) \to \mathbb{S}$ 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 (DWTA) [4, 23, 6, 11] is a tuple $\mathscr{A} = (Q, \Sigma, \delta, \mathrm{wt}, F)$ with

- a finite set Q of states,
- a ranked alphabet Σ of input symbols such that $\Sigma \cap Q = \emptyset$,
- a transition mapping $\delta \colon \Sigma(Q) \to Q$,[2]
- a transition weight assignment $\mathrm{wt} \colon \Sigma(Q) \to \underline{\mathbb{S}}$, and
- a set $F \subseteq Q$ of final states.

The transition and transition weight mappings 'δ' and 'wt' naturally extend to mappings $\hat{\delta} \colon T_\Sigma(Q) \to Q$ and $\hat{\mathrm{wt}} \colon T_\Sigma(Q) \to \underline{\mathbb{S}}$ by

$$
\begin{aligned}
\hat{\delta}(q) &= q & \hat{\delta}(\sigma(t_1, \dots, t_n)) &= \delta(\sigma(\hat{\delta}(t_1), \dots, \hat{\delta}(t_n))) \\
\hat{\mathrm{wt}}(q) &= 1 & \hat{\mathrm{wt}}(\sigma(t_1, \dots, t_n)) &= \mathrm{wt}(\sigma(\hat{\delta}(t_1), \dots, \hat{\delta}(t_n))) \cdot \prod_{i \in [n]} \hat{\mathrm{wt}}(t_i)
\end{aligned}
$$

for every $q \in Q$, $n \in \mathbb{N}$, $\sigma \in \Sigma_n$, and $t_1, \dots, t_n \in T_\Sigma(Q)$. Since $\hat{\delta}(t) = \delta(t)$ and $\hat{\mathrm{wt}}(t) = \mathrm{wt}(t)$ for all $t \in \Sigma(Q)$, we can safely omit the hat and simply write δ and wt for $\hat{\delta}$ and $\hat{\mathrm{wt}}$, respectively. The DWTA $\mathscr{A}$ recognizes the weighted tree language $[\![\mathscr{A}]\!] \colon T_\Sigma \to \mathbb{S}$ such that

$$
[\![\mathscr{A}]\!](t) = \begin{cases} \mathrm{wt}(t) & \text{if } \delta(t) \in F \\ 0 & \text{otherwise} \end{cases}
$$

for all $t \in T_\Sigma$. Two DWTA $\mathscr{A}$ and $\mathscr{B}$ are equivalent if $[\![\mathscr{A}]\!] = [\![\mathscr{B}]\!]$; i.e., their recognized weighted tree languages coincide. A DWTA over the BOOLEAN semifield $\mathbb{B}$ is also called DTA [13, 14] and written (Q, Σ, δ, F) since the component 'wt' 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 DWTA 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 $\varphi\colon T_\Sigma(Q) \to \{0,1\}$ with its support. Finally, the set C_δ of shallow transition contexts is

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

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

For minimization, the weighted (extended) context language of a state is relevant. For every $q \in Q$ the context-semantics $[\![q]\!]_{\mathscr{A}}\colon C_\Sigma(Q) \to \mathbb{S}$ of q is defined for every $c \in C_\Sigma(Q)$ by

$$[\![q]\!]_{\mathscr{A}}(c) = \begin{cases} \mathrm{wt}(c[q]) & \text{if } \delta(c[q]) \in F \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, $[\![q]\!]_{\mathscr{A}}$ is the weighted (extended) language recognized by $\mathscr{A}$ starting in state q. Two states $q, q' \in Q$ are equivalent [5], written $q \equiv q'$, if there exists $s \in \underline{\mathbb{S}}$ such that $[\![q]\!]_{\mathscr{A}}(c) = s \cdot [\![q']\!]_{\mathscr{A}}(c)$ for all $c \in C_\Sigma$. An equivalence relation $\cong \subseteq Q \times Q$ is a congruence relation (for the DWTA $\mathscr{A}$) if for all $n \in \mathbb{N}$, $\sigma \in \Sigma_n$, and $q_1 \cong q'_1, \dots, q_n \cong q'_n$ we have $\delta(\sigma(q_1,\dots,q_n)) \cong \delta(\sigma(q'_1,\dots,q'_n))$. It is known [5] that $\equiv$ is a congruence relation. The DWTA $\mathscr{A}$ is minimal if there is no equivalent DWTA with strictly fewer states. We can compute a minimal DWTA efficiently using a variant of HOPCROFT's algorithm [20, 18] that computes $\equiv$ and runs in time $\mathscr{O}(m \log n)$, where $m = |\Sigma(Q)|$ is the size of $\mathscr{A}$ and $n = |Q|$.

3 A characterization of hyper-minimality

Hyper-minimization [3] is a form of lossy compression that allows any finite number of errors. It has been investigated in [1, 19, 12] for deterministic finite-state automata and in [21] for deterministic tree automata. Finally, hyper-minimization was already generalized to weighted deterministic finite-state automata in [24], from which we borrow much of the general approach. In the following, let $\mathscr{A} = (Q,\Sigma,\delta,\mathrm{wt},F)$ and $\mathscr{B} = (P,\Sigma,\mu,\mathrm{wt}',G)$ be DWTA over the commutative semifield $\langle \mathbb{S},+,\cdot,0,1\rangle$ 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 [24], 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 [5] to obtain the most general statements. Naturally, a scaling factor is not allowed for the almost-equivalence of DWTA since these are indeed supposed assign a different weight to only finitely many trees.

Definition 1. *Two weighted tree languages $\varphi_1, \varphi_2\colon T_\Sigma(Q) \to \mathbb{S}$ are* almost-equivalent*, written $\varphi_1 \approx \varphi_2$, if there exists $s \in \underline{\mathbb{S}}$ such that $\varphi_1(t) = s \cdot \varphi_2(t)$ for almost all $t \in T_\Sigma(Q)$.*[3] *We write $\varphi_1 \approx \varphi_2\ (s)$ to indicate the factor s. The DWTA $\mathscr{A}$ and $\mathscr{B}$ are almost-equivalent if $[\![\mathscr{A}]\!] \approx [\![\mathscr{B}]\!]\ (1)$. Finally, the states $q \in Q$ and $p \in P$ are almost-equivalent if there exists $s \in \underline{\mathbb{S}}$ such that $[\![q]\!]_{\mathscr{A}}(c) = s \cdot [\![p]\!]_{\mathscr{B}}(c)$ 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 DWTA. 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.

[3] "Almost all" means all but a finite number, as usual.

Lemma 2. *Almost-equivalence is an equivalence relation such that $\delta(c[q]) \approx \mu(c[p])$ for all $c \in C_\Sigma$, $q \in Q$, and $p \in P$ with $q \approx p$.*

Proof. Trivially, $\approx$ is reflexive and symmetric (because we have multiplicative inverses for all elements of $\underline{\mathbb{S}}$). Let $\varphi_1, \varphi_2, \varphi_3 \colon T_\Sigma(Q) \to \mathbb{S}$ be weighted tree languages such that $\varphi_1 \approx \varphi_2 \;\; (s)$ and $\varphi_2 \approx \varphi_3 \;\; (s')$ for some $s, s' \in \underline{\mathbb{S}}$. Then there exist finite sets $L, L' \subseteq T_\Sigma(Q)$ such that $\varphi_1(t) = s \cdot \varphi_2(t)$ and $\varphi_2(t') = s' \cdot \varphi_3(t')$ for all $t \in T_\Sigma(Q) - L$ and $t' \in T_\Sigma(Q) - L'$. Consequently, $\varphi_1(t'') = s \cdot s' \cdot \varphi_3(t'')$ for all $t'' \in T_\Sigma(Q) - (L \cup L')$, 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[4] and states. For the second property, induction allows us to easily prove [6] that

$$[\![q]\!]_{\mathscr{A}}(c'[c]) = \mathrm{wt}(c[q]) \cdot [\![\delta(c[q])]\!]_{\mathscr{A}}(c') \qquad \text{and} \qquad [\![p]\!]_{\mathscr{B}}(c_2[c_1]) = \mathrm{wt}'(c_1[p]) \cdot [\![\mu(c_1[p])]\!]_{\mathscr{B}}(c_2) \quad (\dagger)$$

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

$$[\![\delta(c[q])]\!]_{\mathscr{A}}(c') = \frac{[\![q]\!]_{\mathscr{A}}(c'[c])}{\mathrm{wt}(c[q])} = s \cdot \frac{[\![p]\!]_{\mathscr{B}}(c'[c])}{\mathrm{wt}(c[q])} = s \cdot \frac{\mathrm{wt}'(c[p])}{\mathrm{wt}(c[q])} \cdot [\![\mu(c[p])]\!]_{\mathscr{B}}(c')$$

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

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 $\mathscr{A}$ be minimal and $q \approx q' \;\; (s)$ for some $s \in \underline{\mathbb{S}}$ and $q, q' \in Q$. Then*

$$[\![q]\!]_{\mathscr{A}}(c) = s \cdot [\![q']\!]_{\mathscr{A}}(c) \qquad (\ddagger)$$

for almost all $c \in C_\Sigma(Q)$.

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

$$[\![q]\!]_{\mathscr{A}}(c) \cdot \prod_{w \in W} \mathrm{wt}(t_{c(w)}) = [\![q]\!]_{\mathscr{A}}(c') = s \cdot [\![q']\!]_{\mathscr{A}}(c') = s \cdot [\![q']\!]_{\mathscr{A}}(c) \cdot \prod_{w \in W} \mathrm{wt}(t_{c(w)}) \;\; ,$$

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

[4]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 $\mathscr{A}$ is to construct an almost-equivalent device $\mathscr{B}$ such that no device is smaller than $\mathscr{B}$ and almost-equivalent to $\mathscr{A}$. In our setting, the devices are DWTA over the ranked alphabet Σ and the commutative semifield $\mathbb{S}$. Since almost-equivalence is an equivalence relation by Lemma 2, we can replace the requirement "almost-equivalent to $\mathscr{A}$" by "almost-equivalent to $\mathscr{B}$" and call a DWTA $\mathscr{B}$ *hyper-minimal* if no (strictly) smaller DWTA is almost-equivalent to it. Then hyper-minimization equates to the computation of a hyper-minimal DWTA $\mathscr{B}$ that is almost-equivalent to $\mathscr{A}$. Let us first investigate hyper-minimality, which was characterized in [3] for the BOOLEAN semifield using the additional notion of a *preamble* state.

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 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 $\mathscr{O}(m)$ with $m = |\Sigma(Q)|$ being the size of the DWTA $\mathscr{A}$.

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 5. *Let $q, q' \in Q$ and $s \in \underline{\mathbb{S}}$ with $q \neq q'$. The s-*weighted merge *of q into q' is the* DWTA $\mathrm{merge}_{\mathscr{A}}(q \xrightarrow{s} q') = (Q - \{q\}, \Sigma, \delta', \mathrm{wt}', F - \{q\})$ *such that for all $t \in \Sigma(Q - \{q\})$*

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

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 $s \in \underline{\mathbb{S}}$ be such that $q \approx q'$ (s). Then $\mathrm{merge}_{\mathscr{A}}(q \xrightarrow{s} q')$ is almost-equivalent to $\mathscr{A}$.*

Proof. Since $q \approx q'$ (s), there exists a finite set $C \subseteq C_\Sigma$ such that $[\![q]\!]_{\mathscr{A}}(c) = s \cdot [\![q']\!]_{\mathscr{A}}(c)$ for all $c \in C_\Sigma - C$. Let $h \geq \max\ \{\mathrm{ht}(c) \mid c \in C\}$ be an upper bound on the height of the contexts of C. Moreover, let $h' \geq \max\ \{\mathrm{ht}(t) \mid t \in \delta^{-1}(q) \cap T_\Sigma\}$ 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 $\mathscr{B} = \mathrm{merge}_{\mathscr{A}}(q \xrightarrow{s} q')$ and consider an arbitrary tree $t \in T_\Sigma$ whose height is at least z. Clearly, showing that $\mathscr{B}(t) = \mathscr{A}(t)$ for all trees t with $\mathrm{ht}(t) \geq z$ proves that $\mathscr{B}$ and $\mathscr{A}$ are almost-equivalent.[5] Let $W = \{w \in \mathrm{pos}(t) \mid \delta(t|_w) = q\}$ be the set of positions of the subtrees that are recognized in state q. Now $\mathrm{wt}'(t|_w) = s \cdot \mathrm{wt}(t|_w)$ for all $w \in W$ because clearly the subtrees $t|_w$ only use states different from q except at the root, where $\mathscr{A}$ switches to q and $\mathscr{B}$ 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 $w_1 \sqsubset \cdots \sqsubset w_m$, in which $\sqsubseteq$ 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

[5] There 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., $\mathrm{ht}(c_1) > h$) 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 [6] of (†) we obtain

$$\mathscr{A}(t) = \mathscr{A}(c_1[t|_{w_1}]) \stackrel{\dagger}{=} \mathrm{wt}(t|_{w_1}) \cdot [\![q]\!]_{\mathscr{A}}(c_1) = \frac{\mathrm{wt}'(t|_{w_1})}{s} \cdot s \cdot [\![q']\!]_{\mathscr{A}}(c_1) = \mathrm{wt}'(t|_{w_1}) \cdot [\![q']\!]_{\mathscr{A}}(c_1)$$
$$= \mathrm{wt}'(t|_{w_1}) \cdot \begin{cases} \mathrm{wt}(c_1[q']) & \text{if } \delta(c_1[q']) \in F \\ 0 & \text{otherwise.} \end{cases}$$

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

$$= \mathrm{wt}'(t|_{w_1}) \cdot \begin{cases} \mathrm{wt}(c_2[t|_{w_2}]) & \text{if } \delta(c_2[t|_{w_2}]) \in F \\ 0 & \text{otherwise.} \end{cases} \qquad = \mathrm{wt}'(t|_{w_1}) \cdot \mathrm{wt}(t|_{w_2}) \cdot [\![q]\!]_{\mathscr{A}}(c_2)$$
$$\stackrel{\ddagger}{=} \mathrm{wt}'(t|_{w_1}) \cdot \frac{\mathrm{wt}'(t|_{w_2})}{s} \cdot s \cdot [\![q']\!]_{\mathscr{A}}(c_2) = \mathrm{wt}'(t|_{w_1}) \cdot \mathrm{wt}'(t|_{w_2}) \cdot [\![q']\!]_{\mathscr{A}}(c_2) \;\; ,$$

which can now be iterated to obtain

$$= \mathrm{wt}'(t|_{w_1}) \cdot \ldots \cdot \mathrm{wt}'(t|_{w_m}) \cdot [\![q']\!]_{\mathscr{A}}(c_m) = \mathrm{wt}'(t|_{w_1}) \cdot \ldots \cdot \mathrm{wt}'(t|_{w_m}) \cdot [\![q']\!]_{\mathscr{B}}(c_m) \stackrel{\dagger}{=} \mathscr{B}(t) \;\; ,$$

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 $\mathscr{A}$ be the minimal DWTA. For the "only if" part, we know by Lemma 6 that the smaller DWTA $\mathrm{merge}_{\mathscr{A}}(q \xrightarrow{s} q')$ is almost-equivalent to $\mathscr{A}$ if $q \approx q'$ (s) and q is a preamble state. For the "if" direction, suppose that $\mathscr{B}$ is almost-equivalent to $\mathscr{A}$ and $|P| < |Q|$.[6] 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 $\mu(t_1) = p = \mu(t_2)$. Consequently, $q_1 = \delta(t_1) \approx \mu(t_1) = p = \mu(t_2) \approx \delta(t_2) = q_2$, 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 $\mathrm{ht}(t_1), \mathrm{ht}(t_2) \geq |Q|^2$. 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 $\mathscr{A} \cdot \mathscr{B}$. Since $\mathscr{A}$ and $\mathscr{B}$ are almost-equivalent, we have

$$\mathrm{wt}(t_1) \cdot [\![q_1]\!]_{\mathscr{A}}(c) \stackrel{\dagger}{=} [\![\mathscr{A}]\!](c[t_1]) = [\![\mathscr{B}]\!](c[t_1]) \stackrel{\dagger}{=} \mathrm{wt}'(t_1) \cdot [\![p]\!]_{\mathscr{B}}(c)$$
$$\mathrm{wt}(t_2) \cdot [\![q_2]\!]_{\mathscr{A}}(c) \stackrel{\dagger}{=} [\![\mathscr{A}]\!](c[t_2]) = [\![\mathscr{B}]\!](c[t_2]) \stackrel{\dagger}{=} \mathrm{wt}'(t_2) \cdot [\![p]\!]_{\mathscr{B}}(c)$$

for almost all $c \in C_\Sigma$ using again the tree variant of (†). 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,

$$\frac{\mathrm{wt}(t_1) \cdot [\![q_1]\!]_{\mathscr{A}}(c)}{\mathrm{wt}'(t_1)} = \frac{\mathrm{wt}(t_2) \cdot [\![q_2]\!]_{\mathscr{A}}(c)}{\mathrm{wt}'(t_2)} \quad \text{and} \quad [\![q_1]\!]_{\mathscr{A}}(c) = s \cdot [\![q_2]\!]_{\mathscr{A}}(c)$$

for all $c \in C_\Sigma$ and $s = \frac{\mathrm{wt}'(t_1) \cdot \mathrm{wt}(t_2)}{\mathrm{wt}'(t_2) \cdot \mathrm{wt}(t_1)}$, which yields $q_1 \equiv q_2$. This contradicts minimality since $q_1 \neq q_2$, which shows that such a DWTA $\mathscr{B}$ 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 $\mathscr{A}$ with n states
Return: an almost-equivalent hyper-minimal DWTA

	$\mathscr{A} \leftarrow \text{MINIMIZE}(\mathscr{A})$	// $\mathscr{O}(m \log n)$
2:	$K \leftarrow \text{COMPUTEKERNEL}(\mathscr{A})$	// $\mathscr{O}(m)$
	$\overline{K} \leftarrow \text{COMPUTECOKERNEL}(\mathscr{A})$	// $\mathscr{O}(m)$
4:	$(\sim, t) \leftarrow \text{COMPUTEALMOSTEQUIVALENCE}(\mathscr{A}, \overline{K})$	// Algorithm 2 — $\mathscr{O}(m \log n)$
	return $\text{MERGESTATES}(\mathscr{A}, K, \sim, t)$	// Algorithm 3 — $\mathscr{O}(m)$

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 $\mathscr{A}$, respectively.

Definition 8. *A state $q \in Q$ is a* co-preamble state *if* $\text{supp}(\llbracket q \rrbracket_{\mathscr{A}})$ *is finite. Otherwise it is a* co-kernel state. *The sets of all co-preamble states and all co-kernel states are $\overline{P}$ and $\overline{K} = Q - \overline{P}$, 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 $\perp$. 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 $\mathscr{A}$ [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 $\mathscr{A}$ 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 $\square$ 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_δ is the set of transition contexts; i.e., transitions with exactly one occurrence of the symbol $\square$. Moreover, for every $q \in Q$, we let c_q be the smallest transition context $c_q \in C_\delta$ such that

$\delta(c_q[q]) \in \overline{K}$, where the total order on C_δ is arbitrary as assumed earlier, but it needs to be consistently used.

Definition 10. *Given $q \in Q$, its* standardized signature *is*

$$\mathrm{Sig}(q) = \left\{ \langle c, \delta(c[q]), \frac{\mathrm{wt}(c[q])}{\mathrm{wt}(c_q[q])} \rangle \;\Big|\; c \in C_\delta,\, \delta(c[q]) \in \overline{K} \right\} \ .$$

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

Lemma 11. *For all $q, q' \in Q$, if $\mathrm{Sig}(q) = \mathrm{Sig}(q')$, 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 $q, q' \in \overline{K}$, and let $c_q \in C_\delta$ be the smallest transition context such that $c_q[q] \in \overline{K}$. Since q' has the same signature, $c_q = c_{q'}$. In addition, let $s = \frac{\mathrm{wt}(c_q[q])}{\mathrm{wt}(c_q[q'])}$. For every $c \in C_\delta$ and $c' \in C_\Sigma$,

$$[\![q]\!]_{\mathscr{A}}(c'[c]) \stackrel{\dagger}{=} \mathrm{wt}(c[q]) \cdot [\![\delta(c[q])]\!]_{\mathscr{A}}(c') \quad \text{and} \quad [\![q']\!]_{\mathscr{A}}(c'[c]) \stackrel{\dagger}{=} \mathrm{wt}(c[q']) \cdot [\![\delta(c[q'])]\!]_{\mathscr{A}}(c') \ .$$

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

$$\begin{aligned} [\![q]\!]_{\mathscr{A}}(c'[c]) &= \frac{\mathrm{wt}(c[q])}{\mathrm{wt}(c_q[q])} \cdot \mathrm{wt}(c_q[q]) \cdot [\![q_c]\!]_{\mathscr{A}}(c') \ = s_c \cdot \mathrm{wt}(c_q[q]) \cdot [\![q_c]\!]_{\mathscr{A}}(c') \\ &= \frac{\mathrm{wt}(c[q'])}{\mathrm{wt}(c_q[q'])} \cdot \mathrm{wt}(c_q[q]) \cdot [\![q_c]\!]_{\mathscr{A}}(c') = s \cdot [\![q']\!]_{\mathscr{A}}(c'[c]) \end{aligned}$$

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 $[\![q]\!]_{\mathscr{A}}$ and $[\![q']\!]_{\mathscr{A}}$ (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 $\mathscr{A}$ be minimal, and let $q \approx q'$ be such that $\mathrm{Sig}(q) \neq \mathrm{Sig}(q')$. Then there exist $r, r' \in Q$ such that $r \neq r'$ and $\mathrm{Sig}(r) = \mathrm{Sig}(r')$.*

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 $w \in \mathrm{pos}(c)$ 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 $\langle c, q_c, s_c \rangle \notin \mathrm{Sig}(r) = \mathrm{Sig}(r')$ for all $q_c \in Q$ and $s_c \in \underline{\mathbb{S}}$. 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 $\delta(c_r[r]) \in \overline{K}$. Since $q \approx q'$ and $\approx$ is a congruence relation by Lemma 2, we have $r \approx r' \ (s)$ for some $s \in \underline{\mathbb{S}}$, which means that $[\![r]\!]_{\mathscr{A}}(c) = s \cdot [\![r']\!]_{\mathscr{A}}(c)$ for almost all $c \in C_\Sigma$. Consequently,

$$\begin{aligned} \mathrm{wt}(c''[r]) \cdot [\![q_{c''}]\!]_{\mathscr{A}}(c) &= s \cdot \mathrm{wt}(c''[r']) \cdot [\![q_{c''}]\!]_{\mathscr{A}}(c) \\ \mathrm{wt}(c_r[r]) \cdot [\![\delta(c_r[r])]\!]_{\mathscr{A}}(c) &= s \cdot \mathrm{wt}(c_r[r']) \cdot [\![\delta(c_r[r])]\!]_{\mathscr{A}}(c) \end{aligned}$$

Algorithm 2 Algorithm computing the almost-equivalence $\approx$ and scaling map f.

Require: minimal DWTA $\mathscr{A}$ and its co-kernel states $\overline{K}$
Return: almost-equivalence $\approx$ as a partition and scaling map $f\colon Q \to \underline{\mathbb{K}}$

for all $q \in Q$ **do**
2: $\quad \pi(q) \leftarrow \{q\}$; $f(q) \leftarrow 1$ // trivial initial blocks
$h \leftarrow \emptyset$; $I \leftarrow Q$ // hash map of type $h\colon \mathrm{Sig} \to Q$
4: **for all** $q \in I$ **do**
$\quad \mathrm{succ} \leftarrow \mathrm{Sig}(q)$ // compute standardized signature using current δ and $\overline{K}$
6: $\quad$ **if** $\textsc{HasValue}(h, \mathrm{succ})$ **then**
$\quad\quad q' \leftarrow \textsc{Get}(h, \mathrm{succ})$ // retrieve state in bucket ‘succ’ of h
8: $\quad\quad$ **if** $|\pi(q')| \geq |\pi(q)|$ **then**
$\quad\quad\quad \textsc{Swap}(q, q')$ // exchange roles of q and q'
10: $\quad\quad I \leftarrow I \cup \{r \in Q - \{q'\} \mid \exists c \in C_\delta\colon\ \delta(c[r]) = q'\}$ // add predecessors of q'
$\quad\quad f(q') \leftarrow \frac{\mathrm{wt}(c_q[q'])}{\mathrm{wt}(c_q[q])}$ // c_q is as in Definition 10
12: $\quad\quad \mathscr{A} \leftarrow \mathrm{merge}_{\mathscr{A}}(q' \overset{f(q')}{\to} q)$ // merge q' into q
$\quad\quad \pi(q) \leftarrow \pi(q) \cup \pi(q')$ // q and q' are almost-equivalent
14: $\quad\quad$ **for all** $r \in \pi(q')$ **do**
$\quad\quad\quad f(r) \leftarrow f(r) \cdot f(q')$ // recompute scaling factors
16: $\quad h \leftarrow \textsc{Put}(h, \mathrm{succ}, q)$ // store q in h under key ‘succ’
return (π, f)

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

$$\frac{\mathrm{wt}(c''[r])}{\mathrm{wt}(c_r[r])} = \frac{s \cdot \mathrm{wt}(c''[r'])}{s \cdot \mathrm{wt}(c_r[r'])} = \frac{\mathrm{wt}(c''[r'])}{\mathrm{wt}(c_r[r'])} \quad .$$

This proves $\mathrm{Sig}(r) = \mathrm{Sig}(r')$ 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 $\mathscr{O}(n \log n)$ 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.

Algorithm 3 Merging almost-equivalent states.

Require: a minimal DWTA $\mathscr{A}$, its kernel states K, its almost-equivalence $\approx$, and a scaling map $f\colon Q \to \underline{\mathbb{S}}$
Return: hyper-minimal DWTA $\mathscr{A}$ that is almost-equivalent to the input DWTA

for all $B \in (Q/\approx)$ **do**
  select $q \in B$ such that $q \in K$ if possible
  for all $q' \in B - K$ **do**
    $\mathscr{A} \leftarrow \mathrm{merge}_{\mathscr{A}}(q' \xrightarrow{\frac{f(q')}{f(q)}} q)$

Proposition 13. *Algorithm 2 can be implemented to run in time $\mathscr{O}(m \log n)$, where $m = |\Sigma(Q)|$ and $n = |Q|$.*

Finally, we need an adjusted merging process that takes the scaling factors into account. When merging one state into another, their mutual scaling factor can be computed from the scaling map by multiplicaton of one scaling factor with the inverse of the other. Therefore, merging (see Algorithm 3) can be implemented in time $\mathscr{O}(n)$, and hyper-minimization (Algorithm 1) can be implemented in time $\mathscr{O}(m \log n)$ in the weighted setting.

Proposition 14. *Our hyper-minimization algorithm can be implemented to run in time $\mathscr{O}(m \log n)$.*

It remains to prove the correctness of our algorithm. To prove the correctness of Algorithm 2, we still need a technical property.

Lemma 15. *Let $q, q' \in Q$ be states with $q \neq q'$ but $\mathrm{Sig}(q) = \mathrm{Sig}(q')$. Moreover, let $\mathscr{B} = \mathrm{merge}_{\mathscr{A}}(q' \xrightarrow{s} q)$ with $s = \frac{f(q')}{f(q)}$, and let $\cong$ be its almost-equivalence (restricted to P). Then $\cong\, =\, \approx \cap (P \times P)$ where $P = Q - \{q'\}$.*

Proof. Let $p_1 \approx p_2$ with $p_1, p_2 \in P$. Let $c = c_\ell[c_{\ell-1}[\cdots[c_1]\cdots]]$ with $c_1, \ldots, c_\ell \in C_\delta$. Then we obtain the runs

$$R_{p_1} = \langle \delta(c_1[p_1]), \delta(c_2[c_1[p_1]]), \cdots, \delta(c[p_1]) \rangle \quad \text{with weight } \mathrm{wt}(c[p_1])$$
$$R_{p_2} = \langle \delta(c_1[p_2]), \delta(c_2[c_1[p_2]]), \cdots, \delta(c[p_2]) \rangle \quad \text{with weight } \mathrm{wt}(c[p_2]).$$

The corresponding runs R'_{p_1} and R'_{p_2} in $\mathscr{B}$ replace every occurrence of q' in both R_{p_1} and R_{p_2} by q. Their weights are

$$\mathrm{wt}'(c[p_1]) = \begin{cases} \mathrm{wt}(c[p_1]) & \text{if } \delta(c[p_1]) \neq q' \\ \mathrm{wt}(c[p_1]) \cdot s & \text{otherwise} \end{cases}$$
$$\mathrm{wt}'(c[p_2]) = \begin{cases} \mathrm{wt}(c[p_2]) & \text{if } \delta(c[p_2]) \neq q' \\ \mathrm{wt}(c[p_2]) \cdot s & \text{otherwise.} \end{cases}$$

Since $\delta(c'[p_1]) = \delta(c'[p_2])$ for suitably tall contexts $c' \in C_\Sigma$ and $p_1 \approx p_2$, we obtain that $p_1 \cong p_2$. The same reasoning can be used to prove the converse. $\square$

Theorem 16. *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* DWTA *in time* $\mathscr{O}(m \log n)$*, where* $m = |\Sigma(Q)|$ *and* $n = |Q|$.

References

[1] Andrew Badr (2008): *Hyper-Minimization in* $O(n^2)$. In: *Proc. 13th CIAA*, LNCS 5148, Springer, pp. 223–231, doi:10.1007/978-3-540-70844-5_23.

[2] Andrew Badr (2009): *Hyper-Minimization in* $O(n^2)$. *Int. J. Found. Comput. Sci.* 20(4), pp. 735–746, doi:10.1142/S012905410900684X.

[3] Andrew Badr, Viliam Geffert & Ian Shipman (2009): *Hyper-minimizing minimized deterministic finite state automata*. *RAIRO Theor. Inf. Appl.* 43(1), pp. 69–94, doi:10.1051/ita:2007061.

[4] Jean Berstel & Christophe Reutenauer (1982): *Recognizable Formal Power Series on Trees*. *Theor. Comput. Sci.* 18(2), pp. 115–148, doi:10.1016/0304-3975(82)90019-6.

[5] Björn Borchardt (2003): *The Myhill-Nerode Theorem for Recognizable Tree Series*. In: *Proc. 7th DLT*, LNCS 2710, Springer, pp. 146–158, doi:10.1007/3-540-45007-6_11.

[6] Björn Borchardt (2005): *The Theory of Recognizable Tree Series*. Ph.D. thesis, Technische Universität Dresden.

[7] Walter S. Brainerd (1968): *The Minimalization of Tree Automata*. *Information and Control* 13(5), pp. 484–491, doi:10.1016/S0019-9958(68)90917-0.

[8] Cezar Câmpeanu, Nicolae Santean & Sheng Yu (2001): *Minimal cover-automata for finite languages*. *Theor. Comput. Sci.* 267(1–2), pp. 3–16, doi:10.1016/S0304-3975(00)00292-9.

[9] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert & Robert Endre Tarjan (1994): *Dynamic Perfect Hashing: Upper and Lower Bounds*. *SIAM J. Comput.* 23(4), pp. 738–761, doi:10.1137/S0097539791194094.

[10] Jason Eisner (2003): *Simpler and More General Minimization for Weighted Finite-State Automata*. In: *Proc. HLT-NAACL*, The The Association for Computational Linguistics, pp. 64–71.

[11] Zoltán Fülöp & Heiko Vogler (2009): *Weighted tree automata and tree transducers*. In Manfred Droste, Werner Kuich & Heiko Vogler, editors: *Handbook of Weighted Automata*, chapter IX, EATCS Monographs on Theoret. Comput. Sci., Springer, pp. 313–403, doi:10.1007/978-3-642-01492-5_9.

[12] Paweł Gawrychowski & Artur Jeż (2009): *Hyper-minimisation Made Efficient*. In: *Proc. 34th MFCS*, LNCS 5734, Springer, pp. 356–368, doi:10.1007/978-3-642-03816-7_31.

[13] Ferenc Gécseg & Magnus Steinby (1984): *Tree Automata*. Akadémiai Kiadó, Budapest.

[14] Ferenc Gécseg & Magnus Steinby (1997): *Tree Languages*. In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages*, chapter 1, 3, Springer, pp. 1–68, doi:10.1007/978-3-642-59126-6_1.

[15] Jonathan S. Golan (1999): *Semirings and their Applications*. Kluwer Academic, Dordrecht, doi:10.1007/978-94-015-9333-5.

[16] David Gries (1973): *Describing an Algorithm by Hopcroft*. *Acta Inform.* 2(2), pp. 97–109, doi:10.1007/BF00264025.

[17] Udo Hebisch & Hanns J. Weinert (1998): *Semirings — Algebraic Theory and Applications in Computer Science*. World Scientific, doi:10.1142/9789812815965_0001.

[18] Johanna Högberg, Andreas Maletti & Jonathan May (2009): *Backward and Forward Bisimulation Minimization of Tree Automata*. *Theor. Comput. Sci.* 410(37), pp. 3539–3552, doi:10.1016/j.tcs.2009.03.022.

[19] Markus Holzer & Andreas Maletti (2010): *An $n\log n$ Algorithm for Hyper-Minimizing a (Minimized) Deterministic Automaton*. Theor. Comput. Sci. 411(38–39), pp. 3404–3413, doi:10.1016/j.tcs.2010.05.029.

[20] John E. Hopcroft (1971): *An $n\log n$ Algorithm for Minimizing States in a Finite Automaton*. In: Theory of Machines and Computations, Academic Press, pp. 189–196.

[21] Artur Jeż & Andreas Maletti (2013): *Hyper-minimization for deterministic tree automata*. Int. J. Found. Comput. Sci. 24(6), pp. 815–830, doi:10.1142/S0129054113400200.

[22] Dexter Kozen (1992): *On the Myhill-Nerode theorem for trees*. Bulletin of the EATCS 47, pp. 170–173.

[23] Werner Kuich (1998): *Formal Power Series over Trees*. In: Proc. 3rd DLT, Aristotle University of Thessaloniki, pp. 61–101.

[24] Andreas Maletti & Daniel Quernheim (2011): *Hyper-minimisation of deterministic weighted finite automata over semifields*. In: Proc. 13th AFL, Nyíregyháza College, pp. 285–299.

[25] Andreas Maletti & Daniel Quernheim (2011): *Pushing for Weighted Tree Automata*. In: Proc. 36th MFCS, LNCS 6907, Springer, pp. 460–471, doi:10.1007/978-3-642-22993-0_42.

[26] Mehryar Mohri (1997): *Finite-State Transducers in Language and Speech Processing*. Comput. Linguist. 23(2), pp. 269–311.

[27] Slav Petrov, Leon Barrett, Romain Thibaux & Dan Klein (2006): *Learning Accurate, Compact, and Interpretable Tree Annotation*. In: Proc. 44th ACL, The Association for Computational Linguistics, pp. 433–440, doi:10.3115/1220175.1220230.

[28] Daniel Quernheim (2010): *Hyper-minimisation of weighted finite automata*. Master's thesis, Institut für Linguistik, Universität Potsdam.

[29] Robert Endre Tarjan (1972): *Depth-First Search and Linear Graph Algorithms*. SIAM J. Comput. 1(2), pp. 146–160, doi:10.1137/0201010.

[30] Antti Valmari & Petri Lehtinen (2008): *Efficient Minimization of DFAs with Partial Transition Functions*. In: Proc. 25th STACS, LIPIcs 1, Schloss Dagstuhl — Leibniz-Zentrum für Informatik, Germany, pp. 645–656, doi:10.4230/LIPIcs.STACS.2008.1328.