Better Hyper-minimization
Not as Fast, But Fewer Errors

Andreas Maletti*

Departament de Filologies Romàniques, Universitat Rovira i Virgili
Avinguda de Catalunya 35, 43002 Tarragona, Spain
andreas.maletti@urv.cat

Abstract. Hyper-minimization aims to compute a minimal deterministic finite automaton (DFA) that recognizes the same language as a given DFA up to a finite number of errors. Algorithms for hyper-minimization that run in time $O(n \log n)$, where n is the number of states of the given DFA, have been reported recently in [Gawrychowski and Jeż: Hyper-minimisation made efficient. Proc. Mfcs, Lncs 5734, 2009] and [Holzer and Maletti: An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. *Theor. Comput. Sci.* 411, 2010]. These algorithms are improved to return a hyper-minimal DFA that commits the least number of errors. This closes another open problem of [Badr, Geffert, and Shipman: Hyper-minimizing minimized deterministic finite state automata. Rairo *Theor. Inf. Appl.* 43, 2009]. Unfortunately, the time complexity for the obtained algorithm increases to $O(n^2)$.

1 Introduction

Although nondeterministic and deterministic finite automata (NFA and DFA, respectively) are equally expressive [15], NFA can be exponentially smaller than DFA [12, 14], where the size is measured by the number of states. NFA and DFA are used in a vast number of applications that require huge automata like speech processing [13] or linguistic analysis [11]. Consequently, minimization of automata was studied early on. The minimization problem for DFA (NFA) is the computation of an equivalent DFA (NFA) that has the minimal size (i.e., number of states) of all equivalent DFA (NFA). On the bright side, it was shown that DFA can be efficiently minimized in time $O(n \log n)$ [9], where n is the size of the given DFA. However, minimization for NFA is Pspace-complete [10] and thus impractical.

Here we focus on DFA. Although, they can be efficiently minimized, it is often desirable (or even necessary) to sacrifice correctness to minimize further. This leads to the area of lossy compression, in which certain errors are tolerated in order to allow even smaller DFA. A particularly simple error profile is studied in hyper-minimization [1–3, 5–7], where any finite number of errors is allowed. The algorithms of [5–7] run in time $O(n \log n)$, and thus, are asymptotically as efficient as classical minimization. Given a DFA M, they both return a DFA that

* The author was supported by the *Ministerio de Educación y Ciencia* (MEC) grants JDCI-2007-760 and MTM-2007-63422.

M. Domaratzki and K. Salomaa (Eds.): CIAA 2010, LNCS 6482, pp. 201–210, 2011.

– recognizes the same language as M up to a finite number of errors, and
– is minimal among all DFA with the former property (hyper-minimal).

Further, Gawrychowski and Jeż [5] report an algorithm that disallows errors on strings exceeding a specified length. This restriction yields a slightly stricter error profile, but their minimization algorithm still runs in time $O(n \log n)$.

In this paper, we extend the basic hyper-minimization algorithms such that, in addition, the returned DFA commits the least number of errors among all DFA with the two, already mentioned properties. A DFA with those three properties is called 'hyper-optimal'. Note that hyper-optimality depends on the input DFA (or better: its recognized language). Moreover, we return the number of committed errors as a quality measure. It allows a user to disregard the returned DFA if the number of errors is unacceptably large. Our result is based essentially on a syntactic characterization [3, Theorems 3.8 and 3.9] of hyper-minimal DFA. Two DFA are almost-equivalent if their recognized languages differ on only finitely many strings (note that this corresponds to the first item mentioned earlier). A preamble state is a state that can be reached by only finitely many strings from the initial state of the DFA. All remaining states are kernel states. The characterization [3, Theorems 3.8 and 3.9] states that the kernels (i.e., the part of the automaton consisting of the kernel states) of all hyper-minimal, almost-equivalent DFA are isomorphic. Moreover, the preambles are almost-isomorphic, which means that they are isomorphic up to the finality of the states. This yields, as already pointed out in [3], that two hyper-minimal, almost-equivalent DFA differ in only three aspects: (i) the finality of preamble states, (ii) the transitions from preamble states to kernel states, and (iii) the initial state. Thus, the characterization allows us to easily consider all hyper-minimal, almost-equivalent DFA to find a hyper-optimal one. We thus solve an open problem stated in [3]. Unfortunately, the time complexity for the obtained algorithm is $O(n^2)$. Whether it can be improved to $O(n \log n)$ remains an open problem.

2 Preliminaries

The integers and nonnegative integers are denoted by $\mathbb{Z}$ and $\mathbb{N}$, respectively. If the symmetric difference $(S \setminus T) \cup (T \setminus S)$ is finite, then S and T are almost-equal. For finite sets Σ, also called alphabets, the set of all strings over Σ is Σ^*, of which the empty string is $\varepsilon \in \Sigma^*$. Concatenation of strings is denoted by juxtaposition and the length of the word $w \in \Sigma^*$ is $|w|$. A language L over Σ is a subset of Σ^*. A deterministic finite automaton (for short: DFA) is a tuple $M = (Q, \Sigma, q_0, \delta, F)$, in which Q is a finite set of states, Σ is an alphabet of input symbols, $q_0 \in Q$ is an initial state, $\delta \colon Q \times \Sigma \to Q$ is a transition function, and $F \subseteq Q$ is a set of final states. The transition function δ extends to a mapping $\delta \colon Q \times \Sigma^* \to Q$ as follows: $\delta(q, \varepsilon) = q$ and $\delta(q, \sigma w) = \delta(\delta(q, \sigma), w)$ for every $q \in Q$, $\sigma \in \Sigma$, and $w \in \Sigma^*$. For every $q \in Q$, let $L(M)_q = \{w \in \Sigma^* \mid \delta(q_0, w) = q\}$. The DFA M recognizes the language $L(M) = \bigcup_{q \in F} L(M)_q$.

Two states $p, q \in Q$ are equivalent, denoted by $p \equiv q$, if $\delta(p, w) \in F$ if and only if $\delta(q, w) \in F$ for every word $w \in \Sigma^*$. The DFA M is minimal if it does not

Algorithm 1. Structure of the hyper-minimization algorithm [6, 7]

Require: a DFA M

$M \leftarrow \text{MINIMIZE}(M)$ // HOPCROFT's algorithm; $O(m \log n)$
2. $K \leftarrow \text{COMPUTEKERNEL}(M)$ // compute the kernel states; $O(m)$
$\sim \leftarrow \text{AEQUIVALENTSTATES}(M)$ // compute almost-equivalence; $O(m \log n)$
4. $M \leftarrow \text{MERGESTATES}(M, K, \sim)$ // merge almost-equivalent states; $O(m)$
return M

have equivalent states (i.e., $p \equiv q$ implies $p = q$). The name 'minimal' is justified by the fact that no DFA with (strictly) fewer states recognizes the same language as a minimal DFA. For every DFA $M = (Q, \Sigma, q_0, \delta, F)$ an equivalent minimal DFA can be computed efficiently using HOPCROFT's algorithm [8], which runs in time $O(m \log n)$ where $m = |Q \times \Sigma|$ and $n = |Q|$.

3 Hyper-minimization

Let us quickly recall hyper-minimization from [1–3, 6, 7]. We will follow the presentation of [6, 7]. Hyper-minimization is a form of lossy compression with the goal of compressing minimal DFA further at the expense of a finite number of errors. Two DFA M_1 and M_2 such that $L(M_1)$ and $L(M_2)$ are almost-equal are *almost-equivalent*. Moreover, a DFA M that admits no almost-equivalent DFA with (strictly) fewer states is *hyper-minimal*. Consequently, hyper-minimization [1–3] aims to find an almost-equivalent, hyper-minimal DFA.

In the following, let $M = (Q, \Sigma, q_0, \delta, F)$ be a minimal DFA. Let $m = |Q \times \Sigma|$ be the number of its transitions and $n = |Q|$ be the number of its states.

Definition 1 (cf. [3, Definition 2.2]). *Two states $p, q \in Q$ are* k-equivalent *with $k \in \mathbb{N}$, denoted by $p \sim_k q$, if $\delta(p, w) = \delta(q, w)$ for every $w \in \Sigma^*$ such that $|w| \geq k$. The* almost-equivalence $\sim \subseteq Q \times Q$ *is $\sim = \bigcup_{k \in \mathbb{N}} \sim_k$.*

Both k- and almost-equivalence are equivalence relations. The set $\text{Pre}(M)$ of *preamble states* is $\{q \in Q \mid L(M)_q \text{ is finite}\}$, and $\text{Ker}(M) = Q \setminus \text{Pre}(M)$ is the set of *kernel states*. The contributions [5–7] report hyper-minimization algorithms that run in time $O(m \log n)$. The overall structure of the hyper-minimization algorithm [6, 7] is displayed in Algorithm 1, and MERGESTATES is displayed in Algorithm 2. The merge of $p \in Q$ into $q \in Q$ redirects all incoming transitions of p to q. If $p = q_0$ then q is the new initial state. The finality of q is not changed even if p is final. Clearly, the state p can be deleted after the merge if $p \neq q$.

Theorem 2 ([3, Section 4] and [7, Theorem 13]). *In time $O(m \log n)$ Algorithm 1 returns a hyper-minimal DFA that is almost-equivalent to M.*

4 An Example

In this section, we illustrate the problem that we address in this contribution. Namely, we propose an algorithm that not only returns a hyper-minimal,

Algorithm 2. MERGESTATES: Merge almost-equivalent states [6, 7]

Require: a minimal DFA M, its kernel states K, and its almost-equivalent states $\sim$

for all $B \in (Q/\sim)$ **do**
2. select $q \in B$ with $q \in K$ if $B \cap K \neq \emptyset$ // select $q \in B$, preferably a kernel state
for all $p \in B \setminus K$ **do**
4. merge p into q // merge all preamble states of the block into q
return M

almost-equivalent DFA, but rather one that commits the minimal number of errors among all hyper-minimal, almost-equivalent DFA. Moreover, we return the exact number of errors, and we could also return the error strings (at the expense of an increased run-time). We thus solve an open problem of [3].

Throughout this section, we consider the minimal DFA M of Fig. 1, which is essentially the minimal DFA of [3, Fig. 2] with two new states 0 and 2. We added those states because all hyper-minimal DFA that are almost-equivalent to the original DFA of [3] commit exactly 9 errors. Consequently, the existing algorithms already yield DFA with the minimal number of errors. The two new states 0 and 2, of which 0 is the new initial state, change the situation.

The kernel states of M are $\mathrm{Ker}(M) = \{E, F, I, J, L, M, P, Q, R\}$ and the almost-equivalence (represented as a partition) is

$$\{\{0\}, \{2\}, \{A\}, \{B\}, \{C, D\}, \{E\}, \{F\}, \{G, H, I, J\}, \{L, M\}, \{P, Q\}, \{R\}\} \ .$$

Both $\mathrm{Ker}(M)$ and $\sim$ can be computed with the existing algorithms of [2, 3, 5–7]. The hyper-minimization algorithm of [6, 7] might return the hyper-minimal DFA M_1 of Fig. 2 (left), which is almost-equivalent to M. Another such hyper-minimal, almost-equivalent DFA M_2 is presented in Fig. 2 (right). To the author's knowledge there is no hyper-minimization algorithm that can produce M_2. All known algorithms merge both G and H into one of the almost-equivalent kernel states I and J (see Algorithm 2). For example, M_1 is obtained by merging G and H into I. However, M_2 is obtained by merging H into J and G into I. Now, let us look at the errors that M_1 and M_2 make in comparison to M. The following display lists those errors for M_1 (left) and M_2 (right), of which the specific errors of one of the two DFA are underlined.

$$\begin{aligned}
&\{\underline{aa}, aaa, \underline{aaaaa}, aaabaa,\\
&\ aabb, aabbbaa, abb, abbbaa,\\
&\ babb, babbaa, babbbaa, \underline{bbaaa},\\
&\ \underline{bb}, bba, bbabaa, bbbb, bbbbbaa\}
\end{aligned}
\qquad
\begin{aligned}
&\{aaa, aaabaa\\
&\ aabb, aabbbaa, abb, abbbaa,\\
&\ \underline{bab}, \underline{babaaa}, babb, babbaa, babbbaa,\\
&\ bba, bbabaa, bbbb, bbbbbaa\}
\end{aligned}$$

We observe that M_1 commits 17 errors, whereas M_2 commits only 15 errors. Consequently, there is a qualitative difference in different hyper-minimal, almost-equivalent DFA. To be more precise, the quality of the obtained hyper-minimal DFA depends significantly on how the merges are performed.

Let us take a closer look at the cause of the errors. Since the final state C is merged into the non-final state D to obtain M_1, the combined state D of M_1 is

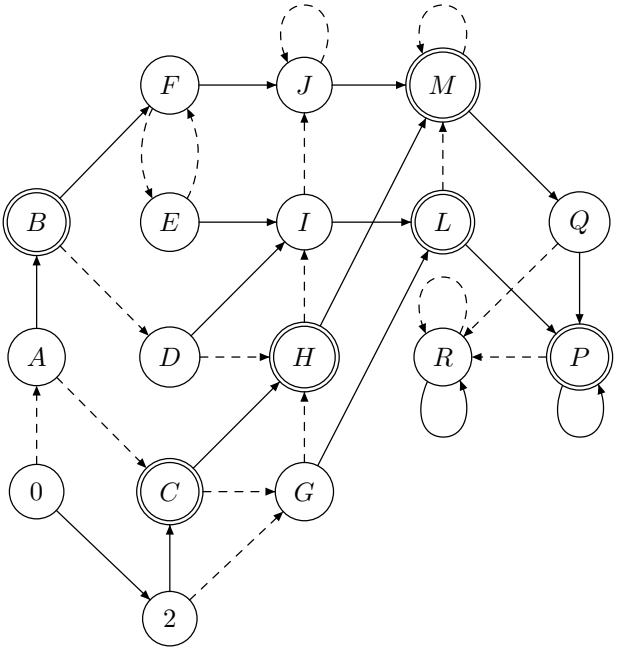

Fig. 1. An example DFA with a-transitions (straight lines) and b-transitions (dashed lines). The initial state is 0.

non-final. Consequently, all strings of $L(M)_C = \{aa, bb\}$, which were accepted in M, are now rejected in M_1. Analogously, the error bab of M_2 is caused by the merge of D into C. The number of those errors can easily be computed with a folklore algorithm (see Algorithm 3 and [4, Lemma 4]) that computes the number of paths from q_0 to each preamble state. Mind that the graph of a DFA restricted to its preamble states is acyclic.

Theorem 3 (see [4, Lemma 4]). *Algorithm 3 computes the number of paths to each preamble state in time $O(m)$.*

Example 4. Algorithm 3 computes the following for M.

$$w(0) = w(2) = w(A) = w(B) = w(D) = 1 \quad w(C) = 2 \quad w(G) = 3 \quad w(H) = 6$$

The remaining errors of M_1 are caused by the merges of G and H into the almost-equivalent kernel state I. Let us denote by $E_{p,q}$ the number of errors made between almost-equivalent states $p \sim q$. More formally, this is the number of strings in the symmetric difference of $L(M_p)$ and $L(M_q)$, where for every $q' \in Q$, the DFA $M_{q'}$ is $(Q, \Sigma, q', \delta, F)$. In other words, $M_{q'}$ is the same DFA as M with initial state q'. Clearly, $E_{q,q} = 0$ and $E_{p,q} = E_{q,p}$ for every $p, q \in Q$. For example, $E_{G,I} = 2$ and $E_{H,I} = 3$ and the corresponding error strings are $\{b, bbaa\}$ and $\{\varepsilon, aa, baa\}$, respectively. Actually, we only need to consider transitions of M_1 that connect preamble to kernel states due to a characterization result of [3]. For

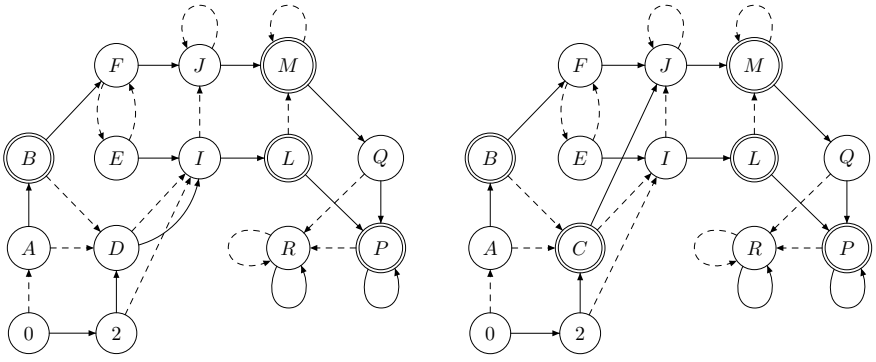

Fig. 2. Two resulting hyper-minimal DFA with a-transitions (straight lines) and b-transitions (dashed lines). The initial state is 0 in both cases.

Algorithm 3. CompAccess: Compute the number of paths to preamble states

Require: a minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$, its preamble states P, and a topological sorting $o\colon \mathbb{N} \to P$ of the preamble states

$w(o(0)) \leftarrow 1$ // the path ε leads to $q_0 = o(0)$

2. **for** $i = 1$ to $|P|$ **do**

$w(o(i)) \leftarrow \sum_{\substack{q \in Q, \sigma \in \Sigma \\ \delta(q,\sigma) = o(i)}} w(q)$ // for each transition (q, σ) leading to $o(i)$ add $w(q)$

4. **return** w

example, for the transition $D \xrightarrow{a} I$ of M_1, we first identify the states of M that were merged into D of M_1. These are C and D of M. Next, we compute the number of paths (in M) to them for each such state q and multiply it with the number of errors made between $\delta(q, a)$ and I. The such obtained error counts are summed up for the total error count. For the three relevant transitions in M_1 we obtain:

$D \xrightarrow{a} I$	$D \xrightarrow{b} I$	$2 \xrightarrow{b} I$
$w(C) \cdot E_{\delta(C,a),I} = 6$	$w(C) \cdot E_{\delta(C,b),I} = 4$	$w(2) \cdot E_{\delta(2,b),I} = 2$
$w(D) \cdot E_{\delta(D,a),I} = 0$	$w(D) \cdot E_{\delta(D,b),I} = 3$	
Sum $= 6$	Sum $= 7$	Sum $= 2$

Thus, we identified $6 + 7 + 2 = 15$ errors. Together with the 2 errors that were caused by the non-finality of D we obtained all 17 errors committed by M_1.

5 Optimal State Merging

The approach presented in the previous section suggests how to compute a hyper-optimal DFA for a given minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ with $m = |Q \times \Sigma|$ and $n = |Q|$. We can simply compute the number of errors for all hyper-minimal, almost-equivalent DFA and select a DFA with a minimal error count. We have

Algorithm 4. CompErrors: Compute the number of errors made between almost-equivalent states

Require: minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ and states $p \sim q$
Global: error matrix $E \in \mathbb{Z}^{Q \times Q}$ initially 0 on the diagonal and -1 everywhere else

if $E_{p,q} \neq -1$ **then**
2. **return** $E_{p,q}$ // if already computed, then return stored value
$c \leftarrow ((p \in F) \text{ xor } (q \in F))$ // set errors to 1 if p and q differ on finality
4. $E_{p,q} \leftarrow c + \sum_{\sigma \in \Sigma} \text{CompErrors}(M, \delta(p,\sigma), \delta(q,\sigma))$ // add errors from follow-states
return $E_{p,q}$ // return the computed value

already seen that different parts (finality and merges) are responsible for the errors. The different parts do not affect each other, which yields that we can compute the number of errors for each choice and greedily select the best one.

First, let us address how to compute the values $E_{p,q}$ for $p \sim q$. Our algorithm is presented in Algorithm 4. It inspects the global matrix E whether the value was already computed. If not, then it checks whether p and q differ on finality (i.e., whether ε is in the symmetric difference of $L(M_p)$ and $L(M_q)$) and adds the error counts $E_{\delta(p,\sigma),\delta(q,\sigma)}$ for each $\sigma \in \Sigma$.

Theorem 5. *Algorithm 4 computes all $E_{p,q}$ with $p \sim q$ in time $O(mn)$.*

Proof. Clearly, the initialization and the recursion for $E_{p,q}$ are correct because each error string w is either the empty string ε or it starts with a letter $\sigma \in \Sigma$. In the latter case, $w' \in \Sigma^*$ with $w = \sigma w'$ is an error string for $E_{\delta(p,\sigma),\delta(q,\sigma)}$. For every $p \sim q$ there exists $k \in \mathbb{N}$ such that $p \sim_k q$ and thus $\delta(p, w) = \delta(q, w)$ for every $w \in \Sigma^*$ with $|w| \geq k$. Consequently, at most k nested recursive calls can occur in the computation of $E_{p,q}$, which proves that the recursion terminates. It remains to prove the time bound. Obviously, if $E_{p,q}$ was already computed, then the algorithm returns immediately. Thus, it makes at most n^2 calls because then all values $E_{p,q}$ are computed. Moreover, there are at most $|\Sigma| + 1$ summands in line 7. Consequently, each call executes in time $O(|\Sigma|)$ apart from the recursive calls, which yields that the algorithm runs in time $O(|\Sigma|n^2) = O(mn)$. □

Example 6. Let us illustrate Algorithm 4 on the example DFA of Fig. 1 and the almost-equivalence $\sim$. We list some error matrix entries together with the corresponding error strings. Note that the error strings are not computed by the algorithm, but are presented for illustrative purposes only.

$E_{Q,P} = 1$	$\{\varepsilon\}$	$E_{H,J} = 2$	$\{\varepsilon, baa\}$	$E_{G,J} = 3$	$\{aa, b, bbaa\}$
$E_{L,M} = 1$	$\{a\}$	$E_{H,I} = 3$	$\{\varepsilon, aa, baa\}$	$E_{G,I} = 2$	$\{b, bbaa\}$
$E_{I,J} = 1$	$\{aa\}$			$E_{G,H} = 5$	$\{\varepsilon, aa, b, baa, bbaa\}$

Next, we need to shortly discuss the structural similarities between hyper-minimal, almost-equivalent DFA. It was shown in [3, Theorems 3.8 and 3.9] that two such DFA

Algorithm 5. CompFinality: Determine finality of a block of preamble states

Require: a minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$, a block of preamble states B, and the number $w(p)$ of access paths for each preamble state p

Global: error count e

$(\overline{f}, f) \leftarrow \Big(\sum_{q \in B \cap F} w(q), \sum_{q \in B \setminus F} w(q) \Big)$ // errors for non-final and final state

2. $e \leftarrow e + \min(\overline{f}, f)$ // add smaller value to global error count

select $q \in B$ such that $q \in F$ if $\overline{f} > f$ // select final state if fewer errors for finality

4. **return** q // return selected state

have isomorphic kernels and almost-isomorphic (by an isomorphism not necessarily respecting finality) preambles. This yields that those DFA only differ on three aspects, which were already identified in [3]:

- the finality of preamble states,
- transitions from preamble states to kernel states, and
- the initial state.

All of the following algorithms will use a global variable e, which will keep track of the number of errors. Initially, it will be set to 0 and each discovered error will increase it. First, we discuss ComputeFinality. For the given block B of almost-equivalent preamble states it computes the number of access paths to final and non-final states in B. Each such path represents a string of $\bigcup_{q \in B} L(M)_q$. After the merge all those strings will take the hyper-minimal DFA into the same state. Thus, making this state final, will cause the number f of errors computed in the algorithm because each access path to a non-final state of B will now access a final state after the merge.

Lemma 7. ComputeFinality(M, B, w) *adds the number of errors made in state p when merging all states of the block B of almost-equivalent preamble states into the state p that is returned by the call.*

Next, we discuss the full merging algorithm (see Algorithm 6). We assume that all values $E_{p,q}$ with $p \sim q$ are already computed. In lines 5–7 we first handle the already discussed decision for the finality of blocks B of preamble states and perform the best merge into state q. In lines 8–11 we investigate the second structural difference between hyper-minimal, almost-equivalent DFA: transitions from preamble to kernel states. Clearly, the preamble state represents a set of exclusively preamble states in the input DFA M and the kernel state represents a set of almost-equivalent states of M that contains at least one kernel state. Consequently, we can simply check whether $\delta(q, \sigma)$ is almost-equivalent to a kernel state. We then consider all almost-equivalent kernel states $q' \sim \delta(q, \sigma)$ and compute the error-count for rerouting the transition to q'. This error count is simply obtained by multiplying the number of paths to a state p in the current block B with the number $E_{\delta(p,\sigma),q'}$ of errors performed between the designated kernel state and the follow-state of the current state. Mind that $\delta(p, \sigma)$ was not

Algorithm 6. OptMerge: Optimal merging of almost-equivalent states

Require: a minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$, its kernel states K, and its almost-equivalent states $\sim$

Global: error count e; initially 0

$P \leftarrow Q \setminus K$ // set P to preamble states
2. $o \leftarrow \text{TopoSort}(P)$ // topological sorting of preamble states; $o\colon \mathbb{N} \to P$
$w \leftarrow \text{CompAccess}(M, P, o)$ // compute the number of access paths for preamble
4. **for all** $B \in (Q/\!\sim)$ such that $B \subseteq P$ **do**
$q \leftarrow \text{CompFinality}(M, B, w)$ // determine finality of merged state
6. **for all** $p \in B$ **do**
merge p into q // perform the merges
8. **for all** $\sigma \in \Sigma$ **do**
if $B' = \{q' \in K \mid q' \sim \delta(q, \sigma)\} \neq \emptyset$ **then**
10. $e \leftarrow e + \min_{q' \in B'} \Big(\sum_{p \in B} w(p) \cdot E_{\delta(p,\sigma),q'} \Big)$ // add best error count
$\delta(q, \sigma) \leftarrow \arg\min_{q' \in B'} \Big(\sum_{p \in B} w(p) \cdot E_{\delta(p,\sigma),q'} \Big)$ // update follow state
12. **if** $B' = \{q' \in K \mid q' \sim q_0\} \neq \emptyset$ **then**
$e \leftarrow e + \min_{q' \in B'} E_{q_0,q'}$ // add best error count
14. $q_0 \leftarrow \arg\min_{q' \in B'} E_{q_0,q'}$ // set best initial state
return (M, e)

affected by the merge in lines 6–7 because the merge only reroutes incoming transitions to p. If there are several states in the current block B, then we sum the obtained error counts. The smallest such error count is then added to the global error count in line 10 and the corresponding designated kernel state is selected as the new target of the transition in line 11. This makes all preamble states that are almost-equivalent to a kernel state unreachable, so they could be removed. Finally, if the initial state is almost-equivalent to a kernel state, then we perform the same steps as previously mentioned to determine the new initial state (i.e., we consider the transition from "nowhere" to the initial state).

A DFA M' is *hyper-optimal for* $L(M)$ if it is hyper-minimal and the cardinality of the symmetric difference between $L(M)$ and $L(M')$ is minimal among all hyper-minimal DFA. Note that a hyper-optimal DFA for $L(M)$ is almost-equivalent to M.

Theorem 8. *Algorithm 6 runs in time $O(mn)$ and returns a hyper-optimal dfa for $L(M)$. In addition, the number of errors committed is returned.*

Proof. The time complexity is easy to check, so we leave it as an exercise. Since the choices (finality, transition target, initial state) are independent, all hyper-minimal, almost-equivalent DFA are considered in Algorithm 6 by [3, Theorems 3.8 and 3.9]. Consequently, we can always select the local optimum for each choice to obtain a global optimum, which proves that the returned number is the

minimal number of errors among all hyper-minimal DFA. Mind that the number of errors would be infinite for a hyper-minimal DFA that is not almost-equivalent to M. Moreover, it is obviously the number of errors committed by the returned DFA, which proves that the returned DFA is hyper-optimal for $L(M)$. □

Corollary 9 (of Theorem 8). *For every* DFA M *we can obtain a hyper-optimal* DFA *for* $L(M)$ *in time* $O(mn)$.

References

1. Badr, A.: Hyper-minimization in $O(n^2)$. In: Ibarra, O.H., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 223–231. Springer, Heidelberg (2008)
2. Badr, A.: Hyper-minimization in $O(n^2)$. Int. J. Found. Comput. Sci. 20(4), 735–746 (2009)
3. Badr, A., Geffert, V., Shipman, I.: Hyper-minimizing minimized deterministic finite state automata. RAIRO Theor. Inf. Appl. 43(1), 69–94 (2009)
4. Eppstein, D.: Finding common ancestors and disjoint paths in DAGs. Tech. Rep. 95-52, University of California, Irvine (1995)
5. Gawrychowski, P., Jeż, A.: Hyper-minimisation made efficient. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 356–368. Springer, Heidelberg (2009)
6. Holzer, M., Maletti, A.: An $n \log n$ algorithm for hyper-minimizing states in a (minimized) deterministic automaton. In: Maneth, S. (ed.) CIAA 2009. LNCS, vol. 5642, pp. 4–13. Springer, Heidelberg (2009)
7. Holzer, M., Maletti, A.: An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. Theor. Comput. Sci. 411(38–39), 3404–3413 (2010)
8. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations, pp. 189–196. Academic Press, London (1971)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison Wesley, Reading (2007)
10. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. 22(6), 1117–1141 (1993)
11. Johnson, C.D.: Formal Aspects of Phonological Description. Monographs on Linguistic Analysis, vol. 3. Mouton, The Hague (1972)
12. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proc. 12th IEEE Annual Symp. Switching and Automata Theory, pp. 188–191. IEEE Computer Society Press, Los Alamitos (1971)
13. Mohri, M.: Finite-state transducers in language and speech processing. Comput. Linguist. 23(2), 269–311 (1997)
14. Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Computers 20(10), 1211–1214 (1971)
15. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3(2), 115–125 (1959)