Notes on Hyper-minimization

Andreas Maletti*

Universität Stuttgart
Institute for Natural Language Processing
Azenbergstraße 12, 70174 Stuttgart, Germany
andreas.maletti@ims.uni-stuttgart.de

Abstract

These notes report on recent advances in the relatively new area of hyper-minimization. Several open questions that were raised in the pioneering article [Badr, Geffert, and Shipman. *Hyper-minimizing minimized deterministic finite state automata.* RAIRO Theor. Inf. Appl., 43(1):69–94, 2009] are addressed here and the solutions, which are mostly taken from the literature, are presented in a uniform style. In particular, the most efficient hyper-minimization algorithms for several error profiles are presented and the languages of hyper-minimal automata are investigated.

1 Overview

Deterministic finite automata (DFA) [30] are a formal model of computation that is simple and enjoys many good algorithmic properties. These qualities led to very successful applications in areas as diverse as speech processing [28], pattern matching [9], and linguistic analysis [21]. In many of its applications huge DFA (i.e., DFA with several million states) are used. Since typically many operations are applied to in a chain to a single DFA, it is important to keep the intermediate results as small as possible, which typically is achieved by minimization [4]. A minimal DFA is a DFA such that all DFAs that recognize the same language are larger, where the size is measured by the number of states. The asymptotically fastest minimization algorithm is due to Hopcroft [17] and runs in time $O(n \log n)$, where n is the size of the input DFA.

In a number of applications the input DFA is derived from noisy data (see, for example, [28]). In these cases, it might be worthwhile to sacrifice the exact preservation of the recognized language in order to achieve better compression (i.e., smaller DFA). Hyper-minimization as introduced in [2] aims to compress DFA beyond the classical notion of 'minimal DFA' at the expense of a finite number

*The author was supported by the German Research Foundation (DFG) grant MA/4959/1–1.

P. Dömösi and Sz. Iván (Eds.): Automata and Formal Languages. AFL 2011
Debrecen, Hungary, August 17–22, 2011. Proceedings, pages 34–49

of errors. Variations such as cover automata minimization [7], which has been explored before hyper-minimization due to its usefulness in compressing finite languages, or k-minimization [11] restrict the length of the error strings instead of their number.

In this survey, we review recent progress in the area of lossy compression for DFA. In particular, we will consider:

- The original notion [2] of hyper-minimization, in which a finite number of errors is allowed. We review the basic notions, recall the properties of hyper-minimal DFA and then prove some basic properties about canonical languages, which are the languages accepted by hyper-minimal DFA. This answers an open question in [2].
- We present the hyper-minimization algorithms that have improved the corresponding algorithms of [2, 1]. The improved algorithms achieve the same asymptotic run-time complexity as HOPCROFT's algorithm for classical DFA minimization. However, all efficient hyper-minimization algorithms are based on a fine-to-coarse approximation, whereas HOPCROFT's algorithm uses a coarse-to-fine (i.e., partition refinement) approximation.
- Another open problem in [2] raised the question, whether we can optimize the obtained hyper-minimal DFA with respect to a secondary criterion. This question makes sense because there is no unique hyper-minimal DFA for a given input language. We recall from the literature that we can optimize the number of errors or the length of the longest error string easily, but that if we want to optimize ratios (for example, saved states vs. errors made), then the problem becomes intractable.
- Finally, we recall cover automata minimization and k-minimization and demonstrate their usefulness in a combination, which is called 'finite-factored DFA' in [1]. Such a DFA can exactly represent certain languages much more succinctly. In particular, we show that the selection of the length bound can be done automatically without any overhead.

These items closely correspond to the structure of the article. A more detailed exposition to each problem including detailed references can be found in the corresponding section. The next section will refresh some very basic notions and can safely be skipped on a first reading.

2 Preliminaries

The set of all integers is $\mathbb{Z}$, and the subset of nonnegative integers is $\mathbb{N}$. The symmetric different between two sets S and T is $S \triangle T = (S \setminus T) \cup (T \setminus S)$. Each finite set Σ is an alphabet, and the set of all strings over it is Σ^*, of which the empty string is ε. The concatenation of two strings $u, v \in \Sigma^*$ is denoted by the juxtaposition uv, and the length of the string $w = a_1 \cdots a_k$ with $a_1, \ldots, a_k \in \Sigma$ is $|w| = k$. Any subset $L \subseteq \Sigma^*$ is a language over Σ.

A deterministic finite automaton (DFA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, 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 mapping, and $F \subseteq Q$ is a set of final states. The transition mapping δ extends to a mapping $\delta\colon Q \times \Sigma^* \to Q$ by $\delta(q, \varepsilon) = q$ 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 $L(q) = \{w \in \Sigma^* \mid \delta(q, w) \in F\}$. Intuitively, $L(q)$ contains all strings that take $\mathcal{A}$ from q into a final state. The DFA $\mathcal{A}$ recognizes the language $L(\mathcal{A}) = L(q_0)$.

Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and $\mathcal{B} = (P, \Sigma, p_0, \mu, G)$ be two DFA. A mapping $h\colon Q \to P$ is a transition homomorphism if $h(\delta(q, \sigma)) = \mu(h(q), \sigma)$ for every $q \in Q$ and $\sigma \in \Sigma$. If additionally $q \in F$ if and only if $h(q) \in G$ for every $q \in Q$, then h is a (DFA) homomorphism. In both cases, h is an isomorphism if it is bijective. We say that the DFA $\mathcal{A}$ and $\mathcal{B}$ are (transition and DFA) isomorphic if there exists a (transition and DFA, respectively) isomorphism $h\colon Q \to P$. The DFA $\mathcal{A}$ and $\mathcal{B}$ are equivalent if $L(\mathcal{A}) = L(\mathcal{B})$. Clearly, (DFA) isomorphic DFA are equivalent. The DFA $\mathcal{A}$ is minimal if there does not exist a DFA with strictly fewer states that recognizes the language $L(\mathcal{A})$. A minimal DFA that is equivalent to $\mathcal{A}$ can be computed efficiently using HOPCROFT's algorithm [16], which runs in time $O(n \log n)$ where $n = |Q|$. Moreover, minimal DFA are equivalent if and only if they are isomorphic.

3 Hyper-minimality and canonical languages

As already mentioned in the Introduction, many applications of regular languages require DFA of enormous sizes. For example, DFA in the area of natural language processing or speech recognition easily have several million states [28]. Minimization [4] of the involved DFA can often help to address this problem, but there are applications in which even the minimal DFA is still too large to handle efficiently. To reduce the size of such DFA further, we can change the model (e.g., cover automata [7], k-entry DFA [23, 15], nondeterministic finite-state automata [30], etc.) or allow errors. The latter approach leads to the area of lossy compression, which was popularized with the 'MPEG-2 Audio Layer III' (MP3) standard [18] for sound files. The MP3 sound file format encodes a (sampled) sound while allowing errors as long as they are outside of the audible spectrum or masked by another sound (for the typical human).

Hyper-minimization [2] of DFA is a form of lossy compression that allows any finite number of errors. We will discuss more refined error profiles in later sections. Formally, two DFA $\mathcal{A}$ and $\mathcal{B}$ are *almost-equivalent* if $L(\mathcal{A}) \triangle L(\mathcal{B})$ is finite. Clearly, almost-equivalence is an equivalence relation. Given a DFA $\mathcal{A}$, hyper-minimization aims to construct an almost-equivalent DFA $\mathcal{B}$ such that no DFA that is smaller than $\mathcal{B}$ is almost-equivalent to $\mathcal{A}$. Since almost-equivalence is an equivalence relation, we can replace the requirement "almost-equivalent to $\mathcal{A}$" by "almost-equivalent to $\mathcal{B}$" and call a DFA $\mathcal{B}$ *hyper-minimal* if no (strictly) smaller DFA is almost-equivalent to it. Then the goal of hyper-minimization becomes the computation of an almost-equivalent hyper-minimal DFA $\mathcal{B}$. Before we proceed with the minimization algorithms in the next section, we first investigate hyper-minimality following roughly the general presentation of [2].

Definition 1 (see [2, Section 2])**.**

- *The languages* $L_1, L_2 \subseteq \Sigma^*$ *are* almost-equal *if* $L_1 \triangle L_2$ *is finite.*
- *The* DFA $\mathcal{A}$ *and* $\mathcal{B}$ *are* almost-equivalent *if* $L(\mathcal{A})$ *and* $L(\mathcal{B})$ *are almost-equal.*
- *The states* $q \in Q$ *and* $p \in P$ *are* almost-equivalent *if* $L(q)$ *and* $L(p)$ *are almost-equal.*

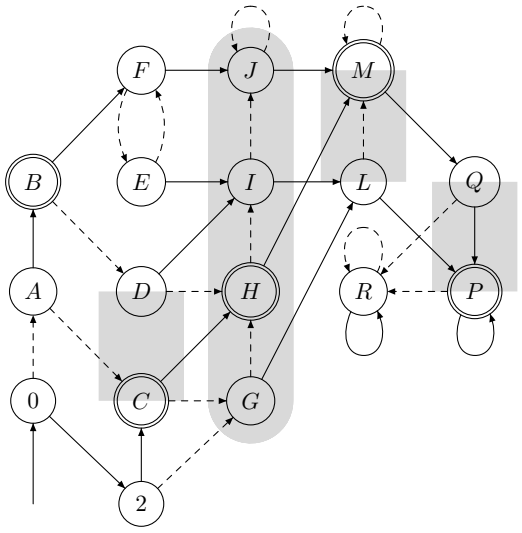

Figure 1: Sample DFA, where the almost-equivalence is indicated. The example is a minor modification (state L is nonfinal here) of [26, Figure 1].

Example 2. *Figure 1 shows a* DFA*, in which we marked almost-equivalence of states. States* L *and* M *are almost-equivalent because*

$$L(L) = \longrightarrow^+ \mid \dashrightarrow^+ \mid \dashrightarrow^+ \longrightarrow \longrightarrow^+$$
$$L(M) = \dashrightarrow^* \mid \dashrightarrow^* \longrightarrow \longrightarrow^+ \ ,$$

which yields that $L(L) \triangle L(M) = \{\varepsilon, \longrightarrow\}$*. The two* DFA *in Figure 2 are almost-equivalent and both almost-equivalent to the* DFA *of Figure 1.*

Almost-equality and almost-equivalence, which are both equivalence relations, are usually denoted by $\sim$ in the following; i.e., $\mathcal{A} \sim \mathcal{B}$ expresses that $\mathcal{A}$ and $\mathcal{B}$ are almost-equivalent. Hyper-minimality was characterized in [2] using the additional notion of a *preamble* state.

Definition 3 (see [2, Definition 2.11])**.** *A state* $q \in Q$ *of* $\mathcal{A}$ *is a* kernel state *if* $\{w \in \Sigma^* \mid \delta(q_0, w) = q\}$ *is infinite. Otherwise, it is a* preamble state*.*

Example 4. *The kernel states of all* DFA *displayed in Figures 1 and 2 are*

$$\{E, F, I, J, L, M, P, Q, R\} \ .$$

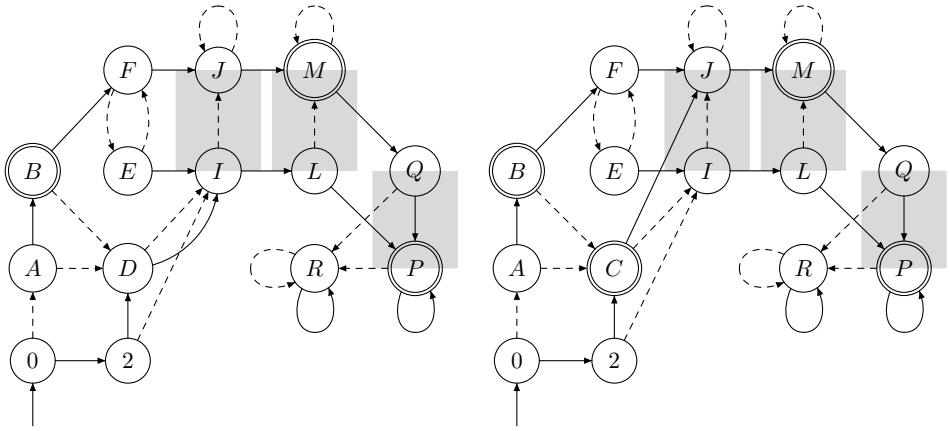

Figure 2: Sample DFA, where the almost-equivalence is indicated. The example is a minor modification (state L is nonfinal here) of [26, Figure 2].

Recall that a (trim) DFA is minimal if and only if it does not have a pair of different, but equivalent states. In analogy, [2] presents a characterization for hyper-minimality.

Theorem 5 (see [2, Theorem 3.4])**.** *A minimal* DFA *is hyper-minimal if and only if it has no pair of different, but almost-equivalent states such that at least of them is a preamble state.*

Example 6. *Using Theorem 5 we can conclude that the* DFA *of Figure 1 is not hyper-minimal because the preamble state G is almost-equivalent to I. In contrast, both* DFA *of Figure 2 are hyper-minimal since $\{I, J, L, M, P, Q\}$ are kernel states in both* DFA *(see Example 4).*

An open question in [2] suggests to call a language $L \subseteq \Sigma^*$ *canonical* if it is recognized by a hyper-minimal DFA. In other words, the language L is canonical if and only if the minimal DFA for L is hyper-minimal. The canonical languages are obviously a proper subset of the recognizable languages [2] because no non-empty finite language is canonical. Moreover, canonical languages are closed under complementation because the complement of a hyper-minimal DFA is still hyper-minimal by Theorem 5. It remained open in [2] which other closure properties canonical languages have.

Theorem 7. *Canonical languages are neither closed under union nor closed under intersection.*

Proof. Consider the DFA $\mathcal{A}$ [left] and $\mathcal{B}$ [right] of Figure 2. We already remarked that they are almost-equivalent and both are hyper-minimal with 14 states. Consequently, $L(\mathcal{A})$ and $L(\mathcal{B})$ are canonical. A simple computation confirms that $L(\mathcal{A}) \cup L(\mathcal{B})$ and $L(\mathcal{A}) \cap L(\mathcal{B})$ are also almost-equivalent to $L(\mathcal{A})$ and $L(\mathcal{B})$. Thus,

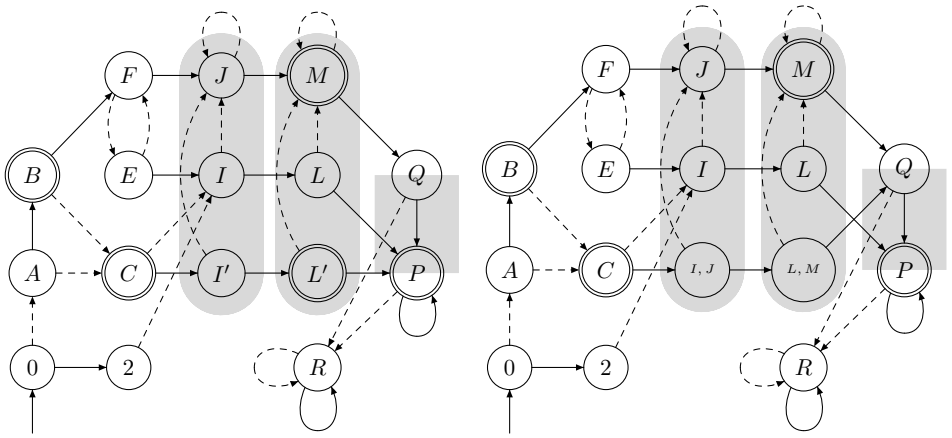

Figure 3: Minimal DFA recognizing the union [left] and the intersection [right] of the languages recognized by the DFA of Figure 2, where almost-equivalence is again indicated.

Algorithm 1 Structure of the hyper-minimization algorithms.

Require: a DFA $\mathcal{A}$
Return: an almost-equivalent hyper-minimal DFA

$\mathcal{A} \leftarrow \textsc{Minimize}(\mathcal{A})$
$K \leftarrow \textsc{ComputeKernel}(\mathcal{A})$
$\sim \leftarrow \textsc{ComputeAlmostEquivalence}(\mathcal{A})$
return $\textsc{MergeStates}(\mathcal{A}, K, \sim)$

the minimal DFA for $L(\mathcal{A}) \cup L(\mathcal{B})$ and $L(\mathcal{A}) \cap L(\mathcal{B})$ should also have 14 states, but it can easily be verified that they have 16 states (see Figure 3). Equivalently, we can observe that I' and L' (the states (I, J) and (L, M), respectively) are preamble states that are almost-equivalent to other states, which proves that both DFA of Figure 3 are not hyper-minimal by Theorem 5. □

As indicated by the naming of states in Figure 3 [left], we need to keep 2 copies of the state L (the states L and L') that differ only in finality for the union. This split also causes the split of the states I and I'. Similarly, we need to keep two nontrivial paired states as seen in Figure 3 [right] for the intersection.

Overall, the author believes that those negative properties render canonical languages rather uninteresting. Consequently, we skip an investigation of other classical closure properties (closure under homomorphism, etc).

4 Hyper-minimization

All hyper-minimization algorithms [2, 1, 11, 14] share the overall structure that is displayed in Algorithm 1. Before we can explain it in detail, we need one more notion. The merge of the state $p \in Q$ into another (i.e., $p \neq q$) state $q \in Q$ redirects all incoming transitions of p to q. If p is the initial state, then q is the new initial state. However, the finality of q is not changed. Since the state p no longer has incoming transitions and no longer is the initial state, it can be deleted after the merge.

Definition 8 (see [27, Section 3]). *For all $p, q \in Q$ with $p \neq q$, we let $\operatorname{merge}_{\mathcal{A}}(p \to q)$ be the* DFA *(P, Σ, p_0, μ, F), where $P = Q \setminus \{p\}$ and for every $r \in Q$ and $\sigma \in \Sigma$*

$$p_0 = \begin{cases} q & \text{if } q_0 = p \\ q_0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mu(r, \sigma) = \begin{cases} q & \text{if } \delta(r, \sigma) = p \\ \delta(r, \sigma) & \text{otherwise.} \end{cases}$$

Lemma 9 (see [27, Lemma 2]). *Let $p, q \in Q$ and $\mathcal{B} = \operatorname{merge}_{\mathcal{A}}(p \to q)$. Then*

$$L(\mathcal{A}) \mathbin{\triangle} L(\mathcal{B}) = \{uw \mid \delta(q_0, u) = p, w \in L(q) \mathbin{\triangle} L(p), \forall u' < u \colon \delta(q_0, u') \neq p\}$$

where $\leq$ is the usual prefix order on Σ^*.

Using Lemma 9 we see that $\mathcal{A}$ and $\operatorname{merge}_{\mathcal{A}}(p \to q)$ are almost-equivalent if (i) q and p are almost-equivalent, which yields that $L(q) \mathbin{\triangle} L(p)$ is finite, and (ii) p is a preamble state, which yields that there are only finitely many $u \in \Sigma^*$ such that $\delta(q_0, u) = p$. All the known hyper-minimization algorithms [2, 1, 11, 14] only perform such merges.

Let us come back to the discussion of the hyper-minimization algorithms. In line 1 of Algorithm 1 we minimize the input DFA using, for example, HOPCROFT's algorithm [16]. In the next step, we compute the set K of kernel states of $\mathcal{A}$ using any algorithm that computes strongly connected components in a directed graph (for example, TARJAN's algorithm [29]). Such an algorithm can be used due to the fact [11, 14] that a state is a kernel state if and only if it is reachable from (i) a nontrivial strongly connected component (i.e., a component of at least 2 states) or (ii) a state with a self-loop. Then in line 3 we compute the almost-equivalence on the states Q, which is the most interesting part and also the part where the algorithms [2, 1, 11, 14] differ. Finally, we merge almost-equivalent states in the way suggested by Theorem 5 until the obtained DFA is hyper-minimal. Since each such merge introduces only finitely many errors, the obtained hyper-minimal DFA is trivially almost-equivalent to the input DFA.

The most interesting part is the computation of the almost-equivalence on the states Q. Let $n = |Q|$. The original algorithm of [2] runs in time $\mathcal{O}(n^3)$ and was improved in [1] to run in time $\mathcal{O}(n^2)$. Finally, [11, 14] independently improved the bound to $\mathcal{O}(n \log n)$, which coincides with the well-known bound for classical DFA minimization [16]. All mentioned algorithms that compute the almost-equivalence use the following observation.

Algorithm 2 Algorithm of [14] computing the almost-equivalence $\sim$.

Require: minimal dfa $\mathcal{A}$
Return: the almost-equivalence relation $\sim$ represented as a partition

```
    π(q) ← {q} for all q ∈ Q          // initial block of q contains just q itself
 2: h ← ∅                             // hash map of type h: Q^Σ → Q
    I ← Q                             // current states
 4: while I ≠ ∅ do
      select and remove q from I      // remove state from I
 6:   succ ← ⟨δ(q, σ) | σ ∈ Σ⟩       // compute successors using current δ
      if HasValue(h, succ) then
 8:     p ← Get(h, succ)             // retrieve state in bucket 'succ' of h
        if |π(p)| ≥ |π(q)| then
10:       Swap(p, q)                // exchange roles of p and q
        I ← I ∪ {r ∈ Q \ {p} | ∃σ: δ(r, σ) = p}   // add predecessors of p
12:     A ← merge_A(p → q)          // merge p into q
        π(q) ← π(q) ∪ π(p)          // p and q are almost-equivalent
14:   h ← Put(h, succ, q)           // store q in h under key 'succ'
    return π
```

Lemma 10 (cf. [2, Definition 2.2])**.** *Let $\mathcal{A}$ be minimal. The states $q, p \in Q$ are almost-equivalent if and only if there is $k \in \mathbb{N}$ such that $\delta(q, w) = \delta(p, w)$ for all $w \in \Sigma^*$ with $|w| \geq k$.*

We present the algorithm of [14, Algorithm 4] in Algorithm 2. Roughly speaking, it computes the successor states for each state and stores them into a hash map in order to avoid pairwise comparisons. Once it finds a pair of states with the same successors (i.e., two states that behave the same for all strings of length at least 1), it merges them and forms a new block in the partition that will eventually represent the almost-equivalence. For efficiency reasons, the merging is performed such that the state representing the bigger block survives (see Lines 10 and 12). Since for each state the size of the "losing" block containing it at least doubles, we obtain that each state is considered at most $\log n$ times. Overall, the algorithm runs in time $\mathcal{O}(n \log n)$ [14, Theorem 9], which answered a question raised in [2].

Theorem 11 (see [14, Theorem 13])**.** *Hyper-minimization can be performed in time $\mathcal{O}(n \log n)$.*

5 Hyper-optimization

Another question raised in [2] was whether we can optimize another criterion such as the number of errors or the length of the longest error. We have already remarked that both DFA of Figure 2 are hyper-minimal and almost-equivalent to

the input DFA of Figure 1, but they differ in the errors that they commit relative to the input DFA. More precisely, they commit the following errors, where $a = \longrightarrow$ and $b = \dashrightarrow$. The specific errors of only one DFA are underlined.

$$\begin{aligned}\{&baba, baba^2, baba^3, bababa, bababa^2,\\ &a^2b^2, a^2b^3a, a^2b^3a^2, b^4, b^5a, b^5a^2\\ &ab^2, ab^3a, ab^3a^2, \underline{aa}, \underline{bb},\\ &a^3, \underline{a^4}, \underline{a^5}, a^3ba, a^3ba^2,\\ &b^2a, \underline{b^2a^2}, \underline{b^2a^3}, b^2aba, b^2aba^2\}\end{aligned}$$

$$\begin{aligned}\{&baba, baba^2, baba^3, bababa, bababa^2,\\ &a^2b^2, a^2b^3a, a^2b^3a^2, b^4, b^5a, b^5a^2\\ &ab^2, ab^3a, ab^3a^2, \underline{bab},\\ &a^3, a^3ba, a^3ba^2,\\ &b^2a, b^2aba, b^2aba^2, \underline{baba^2}, \underline{baba^3}\}\end{aligned}$$

In summary, the DFA of Figure 2 [left] commits 26 errors (displayed left), whereas the DFA of Figure 2 [right] commits only 23 errors (displayed right). The 23 errors coincide with the minimal number of errors, so that the DFA of Figure 2 [right] is optimal. Thus, there indeed is a qualitative difference between the different almost-equivalent hyper-minimal DFA. An example in [27] shows that the gap can be significant.

The question asking for a hyper-minimal DFA with the least errors was answered positively in [26], where it was shown that an almost-equivalent hyper-minimal DFA that commits the least number of errors among all such DFA can be computed in time $\mathcal{O}(n^2)$.

Theorem 12 (see [26, Corollary 9])**.** *We can compute an almost-equivalent hyper-minimal* DFA *in time $\mathcal{O}(n^2)$ that commits the least number of errors among all almost-equivalent hyper-minimal* DFA*.*

This result is based on a characterization [2] of the relation between almost-equivalent hyper-minimal DFA. Roughly speaking, such DFA can only differ in 3 aspects:

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

Before reading the next theorem, the reader might want to recall the notions of a transition and a DFA homomorphism from the Preliminaries.

Theorem 13 (see [2, Theorem 3.9])**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be almost-equivalent hyper-minimal* DFA*. Then there exists a mapping $h\colon Q \to P$ such that*

- *$q \sim h(q)$ for every $q \in Q$,*
- *h yields a transition isomorphism between preamble states of $\mathcal{A}$ and $\mathcal{B}$, and*
- *h yields a* DFA *isomorphism between kernel states of $\mathcal{A}$ and $\mathcal{B}$.*

The simple approach in [26] works because (i) all three aspects are responsible for different errors and (ii) the number of errors introduced in a single merge can easily be computed. However, in contrast to the existing hyper-minimization algorithms the procedure of [26] uses transition merges (rerouting a single transition) instead of state merges (rerouting all incoming transitions). Whether a preamble state should be final or nonfinal can simply be determined by computing

Algorithm 3 CompE(q, p): Compute $|L(q) \triangle L(p)|$.

Require: minimal DFA $\mathcal{A}$ and almost-equivalent states $q \sim p$
Global: error matrix $E \in \mathbb{Z}^{Q \times Q}$ initially 0 on diagonal and -1 elsewhere

if $E_{q,p} = -1$ **then**
  $c \leftarrow (q \in F)$ xor $(p \in F)$ // 1 error if q and p differ on finality
  $E_{q,p} \leftarrow c + \sum_{\sigma \in \Sigma} \text{CompE}(\delta(q, \sigma), \delta(p, \sigma))$ // recursive calls
return $E_{q,p}$ // return the computed value

the number of errors in each case. To decide the target of a rerouted transition from a preamble state to a kernel state, we need to compute the number of errors caused by each potential rerouting. This is achieved with the help of Lemma 9. In all relevant cases, $\text{merge}_{\mathcal{A}}(p \to q)$ is called such that $q \sim p$, which yields that $L(q) \triangle L(p)$ is finite, and p is a preamble state, which yields that there are only finitely many $u \in \Sigma^*$ such that $\delta(q_0, u) = p$. The number of strings that take $\mathcal{A}$ into state p can easily be computed, and Algorithm 3 shows how $|L(q) \triangle L(p)|$ can be computed recursively.

Using essentially the same approach we could also compute an almost-equivalent hyper-minimal DFA that has the shortest errors among all such DFA. However, we will see in the next section, how this problem can be solved using existing methods.

An empirical evaluation of hyper-minimization and hyper-optimization on random DFA can be found in [27]. In summary, hyper-minimization is effective on DFA that are easy to minimize. Surprisingly, DFA that are hard to minimize are also hard to hyper-minimize in the sense that only very few states are saved. The same picture also presents itself for hyper-optimization. In those cases, in which many states can be saved, we can avoid almost all errors using the hyper-optimal DFA. Contrary, the few states saved in the difficult instances for both minimization and hyper-minimization cause a large number of unavoidable errors [27, Section 6].

Finally, we might also be interested in optimizing a ratio in order to balance the number of saved states versus the number of committed errors. Given the DFA $\mathcal{A}$ and integers m and s, can we construct a hyper-minimal DFA $\mathcal{B}$ with at most s states that commits at most m errors (i.e., $|L(\mathcal{B}) \triangle L(\mathcal{A})| \leq m$)? Unfortunately, it is shown in [12, Corollary 1] that deciding whether such a DFA exists is NP-complete. Note that without the restriction on the number of states the problem is easily solvable using the presented hyper-optimization.

6 Cover automata and k-minimization

Some applications only require large, but finite languages [28, 3, 25]. Hyper-minimization would simply return the trivial DFA recognizing the empty language or Σ^*, which is clearly not desired. It was realized in [7] that such a language $L \subseteq \Sigma^*$ is best (exactly) represented by a model, the cover automa-

ton [7], that is slightly different from the classical DFA. A cover automaton is a DFA $\mathcal{A}$ together with an integer k. It accepts a string $w \in \Sigma^*$ if and only if (i) the length of the string is at most k and (ii) $\mathcal{A}$ accepts w. This approach yields that the DFA $\mathcal{A}$ need not represent the language L exactly, but rather it can make any error on strings of length at least $k+1$. More generally, given a DFA $\mathcal{A}$ and an integer k, we can ask for minimal DFA $\mathcal{B}$ such that $L(\mathcal{B}) \cap \Sigma^{\leq k} = L(\mathcal{A}) \cap \Sigma^{\leq k}$, where $\Sigma^{\leq k}$ is the set of all strings of Σ^* whose length is at most k. This problem is generally known as the minimization problem for cover automata [7]. Dually, we can ask for a minimal DFA $\mathcal{B}$ such that $L(\mathcal{B}) \cap \Sigma^{\geq k} = L(\mathcal{A}) \cap \Sigma^{\geq k}$, which is known as k-minimization [11].

We will only consider k-minimization in detail here. Similar results hold for cover automata minimization (see [7, 6, 24, 5, 8, 20]). The procedure for k-minimization [11] is based on the equivalence $\sim_k$, which is defined for every $q \in Q$ and $p \in P$ by $q \sim_k p$ if and only if $L(q) \triangle L(p) \subseteq \Sigma^{\leq k}$. The smallest k such that $q \sim_k p$ is also called the *gap* between q and p and written $\operatorname{gap}(q,p)$. The gap between equivalent states is $-\infty$. Intuitively, the gap between q and p is the length of a longest string on which q and p disagree. However, to limit the length of the error strings, we also need to consider the length of the strings that take the DFA into the states q and p. For example, if we only allow errors on strings of length 8 or less, then a gap of 2 between two states q and p yields an error string of length 9 provided that p has a string of length 7 leading to it. Consequently, we need a more refined notion. Let $\operatorname{level}_{\mathcal{A}}(q)$ be the length of a longest string taking the DFA $\mathcal{A}$ into state q. Formally, $\operatorname{level}_{\mathcal{A}}(q) = \sup\{|w| \mid \delta(q_0, w) = q\}$.

Definition 14 (see [11, Section 4.1])**.** *Let $q \in Q$ and $p \in P$. Then q and p are k-similar if and only if*

$$\operatorname{gap}(q,p) + \min(k, \operatorname{level}_{\mathcal{A}}(q), \operatorname{level}_{\mathcal{B}}(p)) \leq k \ .$$

Unfortunately, k-similarity is not an equivalence relation, but only a compatibility relation (reflexive and symmetric, but not necessarily transitive). The complicated part of deciding whether q and p are k-similar is the computation of $\operatorname{gap}(q,p)$. Fortunately, 'gap' behaves like an ultrametric [10], which allows us to represent it in an ultrametric tree [13, 19, 22]. It is shown in [12, Theorem 5] that this ultrametric tree can be computed in time $\mathcal{O}(n \log n)$. Given the ultrametric tree for 'gap', the computation of a k-minimal DFA becomes easy. Essentially, we can treat the compatibility relation like an equivalence relation as long as we select the right representatives for each block. We skip the details here and refer the interested reader to [11, 12]. Overall, k-minimization (as well as the minimization of cover automata) can be performed in time $\mathcal{O}(n \log n)$ [24, 11, 12].

Theorem 15. *Minimization of cover automata as well as k-minimization can be implemented to run in time $\mathcal{O}(n \log n)$.*

This answers another question of [2] positively. Instead of allowing any finite number of errors as in hyper-minimization, we can as well restrict the length of the errors. However, if we combine the restriction on the length of the errors with

a bound on their number, then the minimization problem becomes intractable. More formally, given integers k and m and a DFA $\mathcal{A}$, it is NP-complete to decide whether there exists a k-minimal DFA $\mathcal{B}$ such that $|L(\mathcal{A}) \triangle L(\mathcal{B})| \leq m$ by [12, Corollary 2].

Let us conclude with an application that combines cover automata minimization and k-minimization. We are given the DFA $\mathcal{A}$ and contrary to the theme of the survey we want to keep exactly the original language. Thus, we have to change the model. Here we select *finite-factored* DFA [1], which are a triple $\mathcal{F} = (\mathcal{B}, k, \mathcal{C})$ of two DFA $\mathcal{B}$ and $\mathcal{C}$ over the same alphabet Σ and an integer k. Such a finite-factored DFA accepts the language

$$L(\mathcal{F}) = \{w \in \Sigma^* \mid w \in L(\mathcal{B}) \cap \Sigma^{\leq k} \text{ or } w \in L(\mathcal{C}) \cap \Sigma^{>k}\} \ .$$

In other words, based on the length of string w, the authorative DFA is selected. If w has length at most k, then $\mathcal{B}$ is authorative. Otherwise, $\mathcal{C}$ decides acceptance. Note that our notion here is slightly different from the one presented in [1, Section 3].

As in [1] (albeit with a smaller alphabet) let us consider the language $L = L_1 \cup L_2$ over $\Sigma = \{0, 1, a, b\}$, where

- $L_1 = \{w \in \{0,1\}^* \mid 9 \geq |w|\}$ and
- $L_2 = \{a, b\}^*$.

In other words, L contains all strings of exclusively digits as long as their length is smaller than 9 and all strings of exclusively letters. A minimal DFA recognizing L is depicted in Figure 4. It has 12 states, most of which are needed to perform the counting. We can represent the same language L using a finite-factored DFA with only 6 states. The finite-factored DFA is presented in Figure 5 [left]. Moreover, if we allow sharing in the graphs of the two DFA constituting a finite-factored DFA, then we only need 4 states. The shared version is depicted in Figure 5 [right].

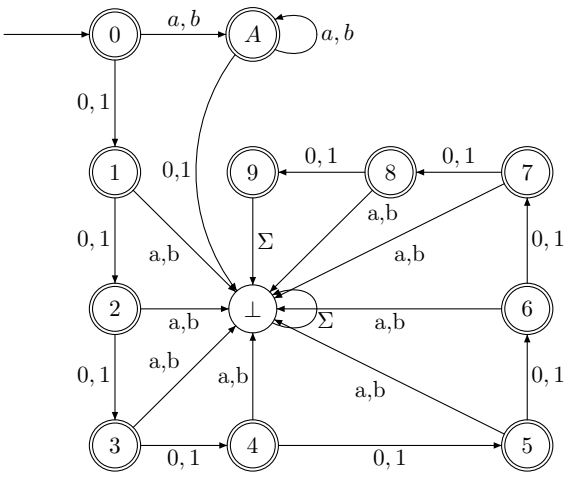

Figure 4: Minimal dfa.

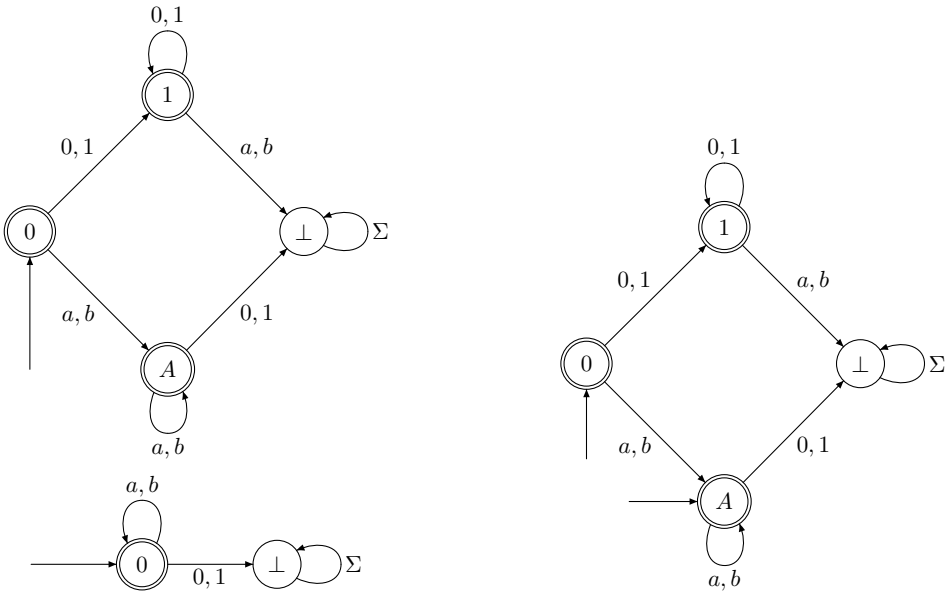

Figure 5: Minimal finite-factored dfa (left: non-shared; right: shared) recognizing the same language as the DFA of Figure 4.

In the previous example, we selected $k = 9$ because it was obviously the best choice for the length split. Then we just minimized the cover automaton $(\mathcal{A}, k)$ and k-minimized $\mathcal{A}$ to obtain $\mathcal{B}$ and $\mathcal{C}$, respectively. Since both $\mathcal{B}$ and $\mathcal{C}$ can be obtained in time $\mathcal{O}(n \log n)$, we can obtain the finite-factored DFA of Figure 5 [left] in time $\mathcal{O}(n \log n)$. Clearly, another choice of k would have yielded different results, which begs the question how to select the optimal value for k. This question was answered in [12, 20]. It is shown that the sizes of the relevant k-minimal DFA and the cover automaton can be computed in time $\mathcal{O}(n \log n)$ for all sensible values of k. These sizes can then be used to compute the optimal value k, or they can be used to compute a hyper-minimal DFA committing the shortest errors as mentioned in the previous section. We obtain the following result, where the size of a finite-factored DFA is simply the sum of the sizes of two constituting DFA.

Theorem 16 (see [12, Theorem 3] and [20, Theorem 11])**.** *We can compute a minimal finite-factored* DFA *recognizing* $L(\mathcal{A})$ *in time* $\mathcal{O}(n \log n)$.

References

[1] Andrew Badr. Hyper-minimization in $O(n^2)$. *Int. J. Found. Comput. Sci.*, 20(4):735–746, 2009.

[2] Andrew Badr, Viliam Geffert, and Ian Shipman. Hyper-minimizing minimized deterministic finite state automata. *RAIRO Theor. Inf. Appl.*, 43(1):69–94, 2009.

[3] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology.* CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, CA, 2003.

[4] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata. Manuscript available at `http://arxiv.org/abs/1010.5318`, 2010.

[5] Cezar Câmpeanu, Andrei Paun, and Jason R. Smith. An incremental algorithm for constructing minimal deterministic finite cover automata. In *Proc. 10th Int. Conf. Implementation and Application of Automata*, volume 3845 of LNCS, pages 90–103. Springer, 2005.

[6] Cezar Câmpeanu, Andrei Paun, and Sheng Yu. An efficient algorithm for constructing minimal cover automata for finite languages. *Int. J. Found. Comput. Sci.*, 13(1):83–97, 2002.

[7] Cezar Câmpeanu, Nicolae Santean, and Sheng Yu. Minimal cover-automata for finite languages. *Theor. Comput. Sci.*, 267(1–2):3–16, 2001.

[8] Jean-Marc Champarnaud, Franck Guingne, and Georges Hansel. Similarity relations and cover automata. *RAIRO Theor. Inf. Appl.*, 39(1):115–123, 2005.

[9] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology.* World Scientific, 2003.

[10] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, second edition, 2002.

[11] Paweł Gawrychowski and Artur Jeż. Hyper-minimisation made efficient. In *Proc. 34th Int. Symp. Mathematical Foundations of Computer Science*, volume 5734 of LNCS, pages 356–368. Springer, 2009.

[12] Paweł Gawrychowski, Artur Jeż, and Andreas Maletti. On minimising automata with errors. Manuscript available at `http://arxiv.org/abs/1102.5682`, 2011.

[13] John A. Hartigan. Representation of similarity matrices by trees. *J. Amer. Statist. Assoc.*, 62(320):1140–1158, 1967.

[14] Markus Holzer and Andreas Maletti. An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. *Theor. Comput. Sci.*, 411(38–39):3404–3413, 2010.

[15] Markus Holzer, Kai Salomaa, and Sheng Yu. On the state complexity of k-entry deterministic finite automata. *J. Autom. Lang. Combin.*, 6(4):453–466, 2001.

[16] John E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[17] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd edition, 2007.

[18] International Organization for Standardization (ISO), Geneva, Switzerland. *ISO/IEC 13818-3*, 2nd edition, 1998.

[19] C. J. Jardine, Nicholas Jardine, and Robin Sibson. The structure and construction of taxonomic hierarchies. *Math. Biosci.*, 1(2):173–179, 1967.

[20] Artur Jeż and Andreas Maletti. Computing all l-cover automata fast. In *Proc. 16th Int. Conf. Implementation and Application of Automata*, LNCS. Springer, 2011.

[21] C. Douglas Johnson. *Formal Aspects of Phonological Description*. Number 3 in Monographs on Linguistic Analysis. Mouton, The Hague, 1972.

[22] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[23] Martin Kappes. Descriptional complexity of deterministic finite automata with multiple initial states. *J. Autom. Lang. Combin.*, 5(3):269–278, 2000.

[24] Heiko Körner. A time and space efficient algorithm for minimizing cover automata for finite languages. *Int. J. Found. Comput. Sci.*, 14(6):1071–1086, 2003.

[25] Cerstin Mahlow and Michael Piotrowski, editors. *State of the Art in Computational Morphology*, volume 41 of *Communications in Computer and Information Science*. Springer, 2009.

[26] Andreas Maletti. Better hyper-minimization — not as fast, but fewer errors. In *Proc. 15th Int. Conf. Implementation and Application of Automata*, volume 6482 of LNCS, pages 201–210. Springer, 2011.

[27] Andreas Maletti and Daniel Quernheim. Optimal hyper-minimization. Manuscript available at `http://arxiv.org/abs/1104.3007`, 2011.

[28] Mehryar Mohri. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311, 1997.

[29] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[30] Sheng Yu. Regular languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2, pages 41–110. Springer, 1997.