On Minimising Automata with Errors*

Paweł Gawrychowski[1,**], Artur Jeż[1,**], and Andreas Maletti[2,***]

[1] Institute of Computer Science, University of Wrocław
ul. Joliot-Curie 15, 50-383 Wrocław, Poland
{gawry,aje}@cs.uni.wroc.pl
[2] Institute for Natural Language Processing, Universität Stuttgart
Azenbergstraße 12, 70174 Stuttgart, Germany
andreas.maletti@ims.uni-stuttgart.de

Abstract. The problem of k-minimisation for a DFA M is the computation of a smallest DFA N (where the size $|M|$ of a DFA M is the size of the domain of the transition function) such that $L(M) \triangle L(N) \subseteq \Sigma^{<k}$, which means that their recognized languages differ only on words of length less than k. The previously best algorithm, which runs in time $\mathcal{O}(|M| \log^2 n)$ where n is the number of states, is extended to DFAs with partial transition functions. Moreover, a faster $\mathcal{O}(|M| \log n)$ algorithm for DFAs that recognise finite languages is presented. In comparison to the previous algorithm for total DFAs, the new algorithm is much simpler and allows the calculation of a k-minimal DFA for each k in parallel. Secondly, it is demonstrated that calculating the least number of introduced errors is hard: Given a DFA M and numbers k and m, it is NP-hard to decide whether there exists a k-minimal DFA N with $|L(M) \triangle L(N)| \leq m$. A similar result holds for hyper-minimisation of DFAs in general: Given a DFA M and numbers s and m, it is NP-hard to decide whether there exists a DFA N with at most s states such that $|L(M) \triangle L(N)| \leq m$.

Keywords: finite automaton, minimisation, lossy compression.

1 Introduction

Deterministic finite automata (DFAs) are one of the simplest devices recognising languages. The study of their properties is motivated by (i) their simplicity, which yields efficient operations, (ii) their wide-spread applications, (iii) their connections to various other areas in theoretical computer science, and (iv) the apparent beauty of their theory. A DFA M is a quintuple $\langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is its finite state-set, Σ is its finite alphabet, $\delta\colon Q \times \Sigma \to Q$ is its partial transition function, $q_0 \in Q$ is its starting state, and $F \subseteq Q$ is its set of accepting

* This work was partially done when A. Maletti was visiting Wrocław University thanks to the support of the "Visiting Professors" programme of the Municipality of Wrocław.
** Supported by MNiSW grant number N N206 492638, 2010–2012.
*** Supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760 and the German Research Foundation (DFG) grant MA/4959/1-1.

F. Murlak and P. Sankowski (Eds.): MFCS 2011, LNCS 6907, pp. 327–338, 2011.

states. The DFA M is total if δ is total. The transition function δ is extended to $\delta\colon Q \times \Sigma^* \to Q$ in the standard way. The language $L(M)$ that is *recognised by the DFA* M is $L(M) = \{w \mid \delta(q_0, w) \in F\}$.

Two DFAs M and N are *equivalent* (written as $M \equiv N$) if $L(M) = L(N)$. A DFA M is *minimal* if all equivalent DFAs are at least as large. One of the classical DFA problems is the *minimisation problem*, which given a DFA M asks for the (unique) minimal equivalent DFA. The asymptotically fastest DFA minimisation algorithm runs in time $\mathcal{O}(|\Sigma|\, n \log n)$ and is due to Hopcroft [7,5], where $n = |Q|$; its variant for partial DFAs is known to run in time $\mathcal{O}(|M| \log n)$.

Recently, minimisation was also considered for hyper-equivalence [2], which allows a finite difference in the languages. Two languages L and L' are *hyper-equivalent* if $|L \triangle L'| < \infty$, where $\triangle$ denotes the symmetric difference of two sets. The DFAs M and N are hyper-equivalent if their recognised languages are. The DFA M is *hyper-minimal* if all hyper-equivalent DFAs are at least as large. The algorithms for hyper-minimisation were gradually improved over time to the currently best run-time $\mathcal{O}(|M| \log^2 n)$ [6,4], which can be reduced to $\mathcal{O}(|M| \log n)$ using a strong computational model (with randomisation or special memory access). Since classical DFA minimisation linearly reduces to hyper-minimisation [6], an algorithm that is faster than $\mathcal{O}(|M| \log n)$ seems unlikely. Moreover, according to the authors' knowledge, randomisation does not help Hopcroft's [3] or any other DFA minimisation algorithm. Thus, the randomised hyper-minimisation algorithm also seems to be hard to improve.

Already [2] introduces a stricter notion of hyper-equivalence. Two languages L and L' are *k-similar* if they only differ on words of length less than k. Analogously, DFAs are k-similar if their recognised languages are. A DFA M is *k-minimal* if all k-similar DFAs are at least as large, and the *k-minimisation problem* asks for a k-minimal DFA that is k-similar to the given DFA M. The known algorithm [4] for k-minimisation of total DFAs runs in time $\mathcal{O}(|M| \log^2 n)$, however it is quite complicated and fails for non-total DFAs.

In this contribution, we present a simpler k-minimisation algorithm for general DFAs, which still runs in time $\mathcal{O}(|M| \log^2 n)$. This represents a significant improvement compared to the complexity for the corresponding total DFA if the transition table of M is sparse. Its running time can be reduced if we allow a stronger computational model. In addition, the new algorithm runs in time $\mathcal{O}(|M| \log n)$ for every DFA M that recognises a finite language. Finally, the new algorithm can calculate (a compact representation of) a k-minimal DFA for each possible k in a single run (in the aforementioned run-time). Outputting all the resulting DFAs might take time $\Omega(n|M| \log^2 n)$.

Although k-minimisation can be efficiently performed, no uniform bound on the number of introduced errors is provided. In the case of hyper-minimisation, it is known [8] that the *optimal* (i.e., the DFA committing the least number of errors) hyper-minimal DFA and the number of its errors m can be efficiently computed. However, this approach does not generalise to k-minimisation. We show the reason. Already the problem of calculating the number m of errors of an optimal k-minimal automaton is NP-hard. Finally, for some applications

it would be beneficial if we could balance the number m of errors against the size $|N|$. Thus, we also consider the question whether given a DFA M and two integers s and m there is a DFA N with at most s states that commits at most m errors (i.e., $|L(M) \triangle L(N)| \leq m$). We show that this problem is also NP-hard.

2 Preliminaries

We usually use the two DFAs $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ and $N = \langle P, \Sigma, \mu, p_0, F'\rangle$. We also write $\delta(w)$ for $\delta(q_0, w)$. The size of DFA M is denoted by $|M|$ and is the number of its non-empty transitions, i.e., entries of δ. The *right-language* $L_M(q)$ of a state $q \in Q$ is the language $L_M(q) = \{w \mid \delta(q, w) \in F\}$ recognised by M starting in state q. Minimisation of DFAs is based on calculating the equivalence $\equiv$ between states, which is defined by $q \equiv p$ if and only if $L_M(q) = L_N(p)$. Similarly, the *left language* of q is the language $\delta^{-1}(q) = \{w \mid \delta(w) = q\}$ of words leading to q in M. For two languages L and L', we define their *distance* $d(L, L')$ as

$$d(L, L') = \min\{\ell \mid L \cap \Sigma^{\geq \ell} = L' \cap \Sigma^{\geq \ell}\} \ ,$$

where $\min \emptyset = \infty$. Actually, d is an ultrametric. The distance d can be extended to states: $d(q, p) = d(L_M(q), L_N(p))$ for $q \in Q$ and $p \in P$. It satisfies the simple recursive formula:

$$d(q, p) = \begin{cases} 0 & \text{if } q \equiv p, \\ 1 + \max\{d(\delta(q, a), \mu(p, a)) \mid a \in \Sigma\} & \text{otherwise.} \end{cases} \tag{1}$$

The minimal DFAs considered in this paper are obtained mostly by state merging. We say that the DFA N is the result of *merging state q to state p* (assuming $q \neq p$) in M if N is obtained from M by changing all transitions ending in q to transitions ending in p and deleting the state q. If q was the starting state, then p is the new starting state. Formally, $P = Q \setminus \{q\}$, $F' = F \setminus \{q\}$, and

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

The process is illustrated in Fig. 1. Let $\text{in-level}_M(q)$ be the length of a longest word leading to q in M. If there is no such longest word, then $\text{in-level}_M(q) = \infty$. Formally, we have $\text{in-level}_M(q) = \sup\{|w| \mid w \in \delta^{-1}(q)\}$ for every $q \in Q$.

3 Efficient k-minimisation

3.1 k-similarity and k-minimisation

Two languages L and L' are *k-similar* if they only differ on words of length smaller than k, and the two DFAs M and N are k-similar if their recognised languages are. The DFA M is *k-minimal* if all k-similar DFAs are at least as large. In this section, we present a general simple algorithm k-Minimise that computes a k-minimal DFA that is k-similar to the input DFA M. Then we present a data structure that allows a fast, yet simple implementation of it.

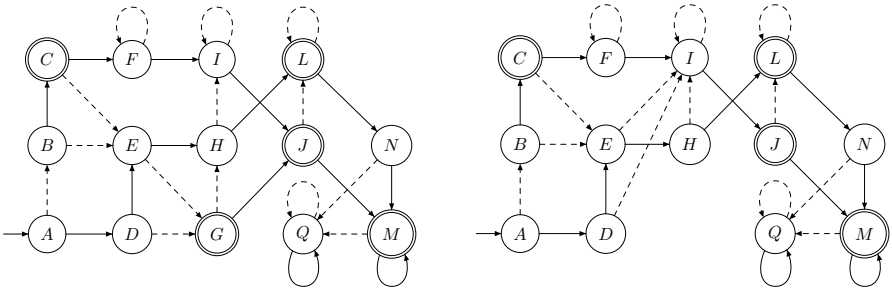

Fig. 1. Merging state G into I

Definition 1. *For two languages L and L', we let $L \sim_k L' \iff d(L, L') \leq k$.*

The hyper-equivalence relation [2] can now be defined as $\sim = \bigcup_k \sim_k$. Next, we extend k-similarity to states.

Definition 2. *Two states $q \in Q$ and $p \in P$ are k-similar, denoted by $q \sim_k p$, if*

$$d(q, p) + \min(k, \text{in-level}_M(q), \text{in-level}_N(p)) \leq k \ .$$

While $\sim_k$ is an equivalence relation on languages, it is, in general, only a compatibility relation (i.e., reflexive and symmetric) on states. On states the hyper-equivalence is not a direct generalisation of k-similarity. Instead, $p \sim q$ if and only if $L_M(q) \sim L_N(p)$. We use the k-similarity relation to give a simple algorithm k-Minimise(M), which constructs a k-minimal DFA (see Alg. 1). In Sect. 3.2 we show how to implement it efficiently.

Theorem 3. k-Minimise *returns a k-minimal DFA that is k-similar to M.*

Proof (sketch). There are two things to show: (i) that the obtained DFA N has the minimal number of states and (ii) that it is k-similar to M. The states of N are pairwise k-inequivalent (when considered in M) and using an approach similar to [4, Lemma 6] it naturally follows that each DFA that is k-similar to M has at least this number of states. For part (ii) we show that after each merge the current DFA N is k-similar to M. To this end, we first show that $\text{in-level}_N(p) \leq \text{in-level}_M(p)$ using a little more general induction hypothesis. Next, we estimate the distance between p regarded as a state in M and in N follows: $d(L_M(p), L_N(p)) \leq k - \text{in-level}_M(p)$. The rest of the proof are simple calculations using that d is an ultrametric. □

3.2 Distance Forests

In this section we define distance forests, which capture the information of the distance between states of a given minimal DFA M. We show that k-minimisation

Alg. 1. k-MINIMISE(M) with minimal M

```
1: calculate ∼_k on Q
2: N ← M
3: while q ∼_k p for some q, p ∈ P and q ≠ p do
4:     if in-level_M(q) ≥ in-level_M(p) then
5:         swap q and p
6:     N ← MERGE(N, q, p)
```

can be performed in linear time, when a distance forest for M is supplied. We start with a total DFA M because in this case the construction is fairly easy. In Sect. 3.3 we show how to extend the construction to non-total DFAs.

Let $\mathcal{F}$ be a forest (i.e., set of trees) whose leaves are enumerated by Q and whose edges are weighted by elements of $\mathbb{N}$. For convenience, we identify the leaf vertices with their label. For every $q \in Q$, we let $\text{tree}(q) \in \mathcal{F}$ be the (unique) tree that contains q. The level 'level(v)' of a vertex v in $t \in \mathcal{F}$ is the maximal weight of all paths from v to a leaf, where the weights are added along a path. Finally, given two vertices v_1, v_2 of the same tree $t \in \mathcal{F}$, the lowest common ancestor of v_1 and v_2 is the vertex $\text{lca}(v_1, v_2)$.

Definition 4 (Distance forest). *Let $\mathcal{F}$ be a forest whose leaves are enumerated by Q. Then $\mathcal{F}$ is a* distance forest *for M if for every $q, p \in Q$ we have*

$$d(q,p) = \begin{cases} \text{level}(\text{lca}(q,p)) & \text{if } \text{tree}(q) = \text{tree}(p), \\ \infty & \text{otherwise.} \end{cases}$$

To construct a distance forest we use (1) to calculate the distance. Since M is minimal, there are no states with distance 0. In phase ℓ, we merge all states at distance exactly ℓ into one state. Since we merged all states of distance at most $\ell - 1$ in the previous phases, we only need to identify the states of distance 1 in the merged DFA. Thus we simply group the states according to their vectors of transitions by letters from $\Sigma = \{a_1, \ldots, a_m\}$. To this end we store these vectors in a dictionary, which we organise as a trie of depth m. The leaf of a trie corresponding to a path $(q_1, \ldots, q_m)$ keeps a list of all states q such that $\delta(q, a_i) = q_i$ for every $1 \le i \le m$. For each node v in the trie we keep a *linear dictionary* that maps a state q into a child of v. We demand that this linear dictionary supports search, insertion, deletion, and enumeration of all elements.

Theorem 5. *Given a total DFA M, we can build a distance forest for M using $\mathcal{O}(|M| \log n)$ linear-dictionary operations.*

We now shortly discuss some possible implementations of the linear dictionary. An implementation using balanced trees would have linear space consumption and the essential operations would run in time $\mathcal{O}(\log n)$. If we allow randomisation, then we can use dynamic hashing. It has a worst-case constant time look-up and an amortised expected constant time for updates [9]. Since it is natural to assume that $\log n$ is proportional to the size of a machine word, we can hash

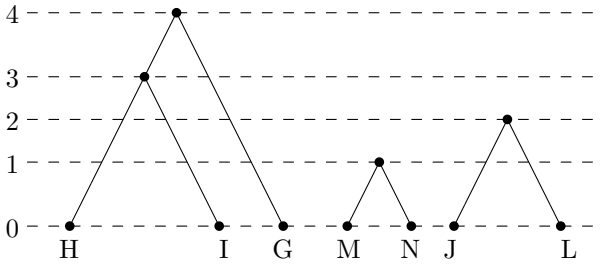

Fig. 2. A distance forest for the left DFA of Fig. 1. Single-node trees are omitted.

in constant time. We can obtain even better time bounds by turning to more powerful models. In the RAM model, we can use exponential search trees [1], whose time per operation is $\mathcal{O}(\frac{(\log\log n)^2}{\log\log\log n})$ in linear space. Finally, if we allow a quadratic space consumption, which is still possible in sub-quadratic time, then we can allocate (but not initialise) a table of size $|M| \times n$. Standard methods can be used to keep track of the actually used table entries, so that we obtain a constant run-time for each operation, but at the expense of $\Theta(|M|\, n)$ space; i.e., quadratic memory consumption.

We can now use a distance forest to efficiently implement k-Minimise. For each state q we locate its highest ancestor v_q with $\text{level}(v_q) \leq k - \text{in-level}(q)$. Then q can be merged into any state that occurs in the subtree rooted in v_q (assuming it has a smaller in-level). This can be done using a depth-first traversal on the trees of the distance forest. A more elaborate construction based on this approach yields the following.

Theorem 6. *Given a distance forest for M, we can compute the size of a k-minimal DFA that is k-similar to M for all k in time $\mathcal{O}(|M|)$. For a fixed k, we can also compute a k-minimal DFA in time $\mathcal{O}(|M|)$. Finally, we can run the algorithm in time $\mathcal{O}(|M| \log n)$ such that it has a k-minimal DFA stored in memory in its k-th phase.*

3.3 Finite Languages and Partial Transition Functions

The construction of a distance forest was based on a total transition function δ, and the run-time was bounded by the size of δ. We now show a modification for the non-total case. The main obstacle is the construction of a distance forest for an acyclic DFA. The remaining changes are relatively straightforward.

Theorem 7. *For every acyclic DFA M we can build a distance forest in time $\mathcal{O}(|M| \log n)$.*

Proof (sketch). Since $L(M)$ is finite, we have that $m(p) = \max\{|w| \,|\, w \in L_M(p)\}$ is a natural number for every state p. Let $Q_i = \{p \,|\, m(p) = i\}$ and $Q_{<\infty} = \bigcup_i Q_i$. Every state has a finite right-language, and thus every distance forest consists of a single tree. We iteratively construct the fragments of this tree by starting from

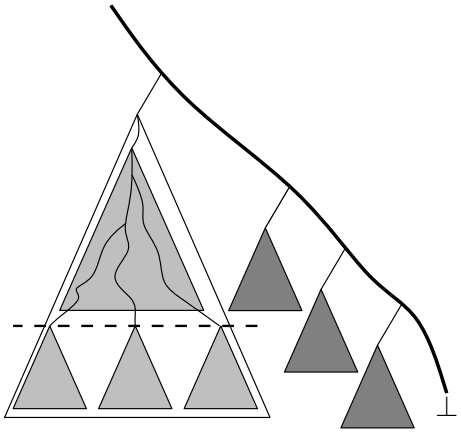

Fig. 3. Illustration for the construction of the distance tree. The spine is depicted using with a thicker line. Splitting one fragment into smaller recursive calls is shown.

a single leaf $\perp$, which represents the empty language and "undefinedness" of the transition function. Before we start to process Q_t, we have already constructed the distance tree for $\bigcup_{i<t} Q_i$. The constructed fragments are connected to a single path, called the *spine*, which ends at the leaf $\perp$ (see Fig. 3).

Let $Q_t = \{p_1, \ldots, p_s\}$, and let $v \in Q_t$. Moreover, let $\mathrm{f}(v)$ be the vector of states $\mathbf{v} = (\delta(v, a))_{a \in \Sigma}$, where the coordinates are sorted by a fixed order on Σ. Define the distance between those vectors as

$$d((p_a)_{a\in\Sigma}, (p'_a)_{a\in\Sigma}) = \max\{d(p_i, p'_i) + 1 \mid a \in \Sigma\} \ ,$$

where we know that $d(p_i, \perp) = m(p_i)$ and $d(\perp, p'_i) = m(p'_i)$. Similarly to the distance, we can define the father $\mathrm{f}(\mathbf{v})$ of a vector $\mathbf{v} = (p_a)_{a\in\Sigma}$ as $\mathrm{f}(\mathbf{v}) = (\mathrm{f}(p_a))_{a\in\Sigma}$. Then

$$\mathrm{f}^{\ell+1}(v) = \mathrm{f}^{\ell+1}(v') \iff \mathrm{f}^{\ell}(\mathbf{v}) = \mathrm{f}^{\ell}(\mathbf{v}').$$

We can now use a divide-and-conquer approach: First, for each vector we calculate its 2^k-th ancestor, where $k = \lceil \log s/2 \rceil$. Then all such vectors are sorted according to their ancestors, in particular they are partitioned into blocks with the same ancestors. After that we recurse onto those (bottom) blocks that have more than two entries and onto the upper block, which consists of the different 2^k-ancestors. The recursion ends for blocks containing at most two vectors, for which we calculate the distance tree directly. □

For every state $q \in Q$, its *signature* $\mathrm{sig}(q)$ is $\{a \mid L_M(\delta(q, a)) \text{ is infinite}\}$. If $\mathrm{sig}(q) \neq \mathrm{sig}(p)$, then $d(q, p) = \infty$, which allows us to keep a separate dictionary for each signature. Let us fix such a trie. To take into account also the transitions by letters outside the signature, we introduce a fresh letter \$, whose transitions are represented in the trie as well. We organize them such that in phase ℓ the \$-transitions for the states q and p are the same if and only if

$\max\{d(\delta(q,a),\delta(p,a)) \mid a \notin \mathrm{sig}(q)\} \leq \ell - 1$. This is easily organised if the distance forest for all states with a finite right-language is supplied.

Theorem 8. *Given a (non-total) DFA M we can build a distance forest for it using $\mathcal{O}(|M| \log n)$ linear-dictionary operations.*

4 Hyper-equivalence and Hyper-minimisation

When considering minimisation with errors, it is natural that one would like to impose a bound on the total number of errors introduced by minimisation. In this section, we investigate whether given $m, s \in \mathbb{N}$ and a DFA M we can construct a DFA N such that:

1. N is hyper-equivalent to M; i.e., $N \sim M$,
2. N has at most s states, and
3. N commits at most m errors compared to M; i.e., $|L(N) \mathbin{\triangle} L(M)| \leq m$.

Let us call the general problem 'error-bounded hyper-minimisation'. We show that this problem is intractable (NP-hard).

To show NP-hardness of the problem we reduce the 3-colouring problem to it. Roughly speaking, we construct the DFA M from a graph $G = \langle V, E\rangle$ as follows. Each vertex $v \in V$ is represented by a state $v \in Q$, and each edge $e \in E$ is represented by a symbol $e \in \Sigma$. We introduce additional states in a way such that their isomorphic copies are present in any minimal DFA that is hyper-equivalent to M. The additional states are needed to ensure that for every edge $e = \{v_1, v_2\} \in E$ the languages $L_M(\delta(v_1, e))$ and $L_M(\delta(v_2, e))$ differ. Now we assume that $m = |E| \cdot (|V| - 2)$ and $s = 14$. We construct the DFA M such that all vertices of $V \subseteq Q$ are hyper-equivalent to each other and none is hyper-equivalent to any other state. We can save $|V| - 3$ states by merging all states of V into at most 3 states. These merges will cause at least $|E| \cdot (|V| - 2)$ errors. Additionally, 3 states will become superfluous after the merges, so that we can save $|V|$ states. There are two cases:

- If the input graph G is 3-colourable by $c\colon V \to [3]$, then we can merge all states of $c^{-1}(i)$ into a single state for every $i \in [3]$. Since c is proper, we never merge states $v_1, v_2 \in Q$ with $\{v_1, v_2\} \in E$, which avoids further errors.
- On the other hand, if G is not 3-colourable, then we merge at least two states $v_1, v_2 \in Q$ such that $e = \{v_1, v_2\} \in E$. This merge additionally introduces 2 errors caused by the difference $L(\delta(v_1, e)) \mathbin{\triangle} L(\delta(v_2, e))$.

Consequently, a DFA that (i) is hyper-equivalent to M, (ii) has at most s states, and (iii) commits at most m errors exists if and only if G is 3-colourable. This shows that error-bounded hyper-minimisation is NP-hard.

Definition 9. *We construct a DFA $M = \langle Q, \Sigma, \delta, \top, F\rangle$ as follows:*

- $Q = \{\top, \bot, \infty, ☺, ☹\} \cup V \cup \{\bigcirc_j \mid \bigcirc \in \{☺, ☻, ☹\}, j \in [3]\}$,
- $\Sigma = \{a, b\} \cup V \cup E$,

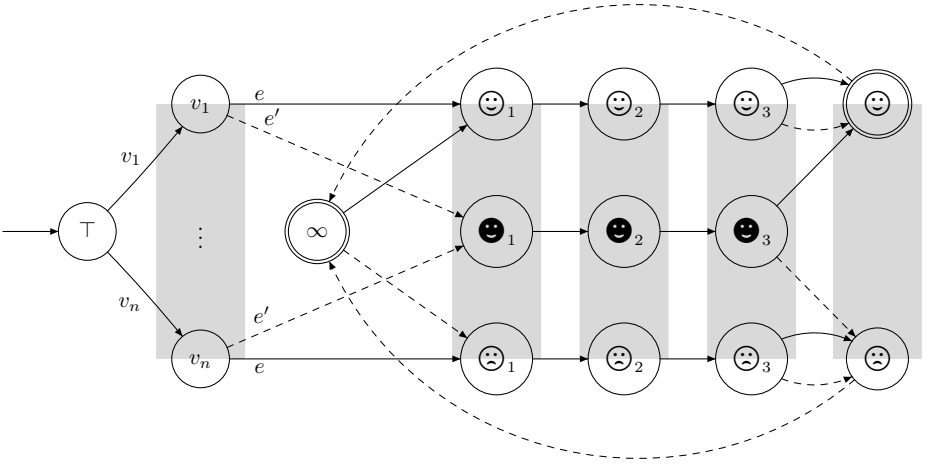

Fig. 4. DFA M constructed in Sect. 4, where a-transitions are represented by unbroken lines (unless noted otherwise), b-transitions by dashed lines, and $e = \{v_1, v_n\}$ and $e' = \{v_2, v_3\}$ with $v_1 < v_2 < v_3 < v_n$. The hyper-equivalence $\sim$ is indicated.

- $F = \{\infty, ☺\}$,
- *for every* $v \in V$, $e = \{v_1, v_2\} \in E$ *with* $v \notin e$ *and* $v_1 < v_2$, $\bigcirc \in \{☺, ☹\}$

$$\delta(\top, v) = v \qquad \delta(\infty, a) = ☺_1 \qquad \delta(\infty, b) = ☹_1$$
$$\delta(v, e) = ●_1 \qquad \delta(v_1, e) = ☺_1 \qquad \delta(v_2, e) = ☹_1$$
$$\delta(●_1, a) = ●_2 \quad \delta(●_2, a) = ●_3 \quad \delta(●_3, a) = ☺ \quad \delta(●_3, b) = ☹$$
$$\delta(\bigcirc_1, a) = \bigcirc_2 \quad \delta(\bigcirc_2, a) = \bigcirc_3 \quad \delta(\bigcirc_3, a) = \bigcirc \quad \delta(\bigcirc_3, b) = \bigcirc \quad \delta(\bigcirc, b) = \infty$$

- *For all remaining cases, we set* $\delta(q, \sigma) = \bot$.

Consequently, the DFA M has $14 + |V|$ states (see Fig. 4). Next, we show how to collapse hyper-equivalent states using a proper 3-colouring $c\colon V \to [3]$ to obtain only 14 states.

Definition 10. *Let* $c\colon V \to [3]$ *be a proper* 3*-colouring for* G. *We construct the DFA* $c(M) = \langle P, \Sigma, \mu, \top, F\rangle$ *where*

- $P = \{\top, \bot, \infty, ☺, ☹\} \cup [3] \cup \{\bigcirc_j \mid \bigcirc \in \{☺, ☹\}, j \in [3]\}$,
- $\mu(p, \sigma) = \delta(p, \sigma)$ *for all* $p \in P \setminus \{\top, 1, 2, 3\}$ *and* $\sigma \in \Sigma$, *and*
- *for every* $v \in V$, $i \in [3]$, *and* $e = \{v_1, v_2\} \in E$ *with* $v_1 < v_2$

$$\mu(\top, v) = c(v) \qquad \mu(i, e) = \begin{cases} ☺_1 & \text{, if } c(v_2) \neq i \\ ☹_1 & \text{, otherwise.} \end{cases}$$

Lemma 11. *There exists a DFA that has at most* 14 *states and commits at most* $|E| \cdot (|V| - 2)$ *errors when compared to* M *if and only if* G *is* 3*-colourable.*

Corollary 12. *'Error-bounded hyper-minimisation' is NP-complete. More formally, given a DFA M and two integers $m, s \in \mathrm{poly}(|M|)$, it is NP-complete to decide whether there is a DFA N with at most s states and $|L(M) \triangle L(N)| \leq m$.*

5 Error-Bounded k-minimisation

In Sect. 3 the number of errors between M and the constructed k-minimal DFA was not calculated. In general, there is no unique k-minimal DFA for M and the various k-minimal DFAs for M can differ in the number of errors that they commit relative to M. Since several dependent merges are performed in the course of k-minimisation, the number of errors between the original DFA M and the resulting k-minimal DFA is not necessarily the sum of the errors introduced for each merging step. This is due to the fact that errors made in one merge may be cancelled out in a subsequent merge. It is natural to ask, whether it is nevertheless possible to *efficiently* construct an *optimal k-minimal* DFA for M (i.e., a k-minimal DFA with the least number of errors introduced). In the following we show that the construction of an optimal k-minimal DFA for M is NP-hard.

The intractability is shown by a reduction from the 3-colouring problem for a graph $G = \langle V, E\rangle$ in a similar, though much more refined, way as in Sect. 4. We again construct a DFA M with one state v for every vertex $v \in V$ and one letter e for each edge $e \in E$. We introduce three additional states $\{1_0, 2_0, 3_0\}$ (besides others) to represent the 3 colours. For the following discussion, let $N = \langle P, \Sigma, \mu, p_0, F'\rangle$ be a k-minimal DFA for M. Let us fix an edge $e = \{v_1, v_2\} \in E$. The DFA M is constructed such that the languages $L_M(\delta(v_1, e))$ and $L_M(\delta(v_2, e))$ have a large but finite symmetric difference; as in the previous section, if a proper 3-colouring $c\colon V \to [3]$ exists the DFA N can be obtained by merging each state v into $c(v)_0$. In addition, for every edge $e = \{v_1, v_2\} \in E$ and vertex $v \in e$, we let $\mu(c(v)_0, e) = \delta(v, e)$. On the other hand, if G admits no proper 3-colouring, then the DFA N is still obtained by state merges performed on M. However, because G has no proper 3-colouring, in the constructed DFA M there exist 2 states v_1, v_2 such that $e = \{v_1, v_2\} \in E$ and that both v_1 and v_2 are merged into the same state $p \in P$. Then the transition $\mu(p, e)$ cannot match both $\delta(v_1, e)$ and $\delta(v_2, e)$. In order to make such an error costly, the left languages of v and v' are designed to be large, but finite. In contrast, we can easily change the transitions of states $\{1_0, 2_0, 3_0\}$ by letters e because the left-languages of the states $\{1_0, 2_0, 3_0\}$ are small.

To keep the presentation simple, we will use two gadgets. The first one will enable us to make sure that two states cannot be merged: k-similar states are also hyper-equivalent, so we can simply avoid undesired merges by making states hyper-inequivalent. Another gadget will be used to increase the in-level of certain states to a desired value.

Lemma 13. *For every congruence $\simeq \subseteq Q \times Q$ on M, there exists a DFA N such that (i) $p_1 \not\sim p_2$ for every $p_1 \in P \setminus Q$ and $p_2 \in P$ with $p_1 \neq p_2$, and (ii) $q_1 \not\sim q_2$ in N for all $q_1 \not\simeq q_2$.*

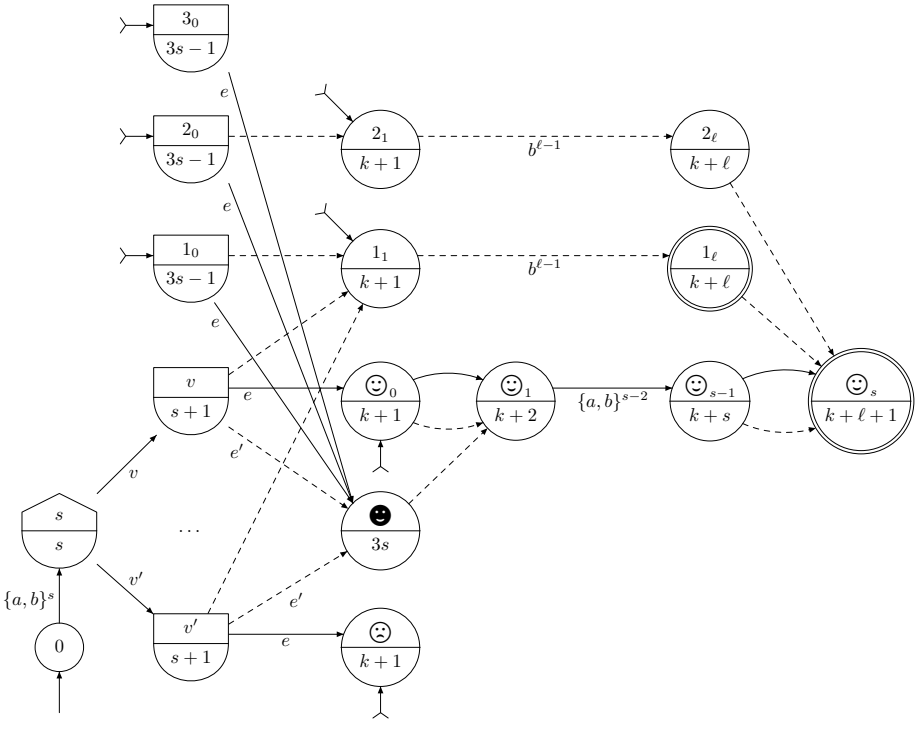

Fig. 5. Illustration of the DFA M of Section 5

In graphical illustrations, we use different shapes for q_1 and q_2 to indicate that $q_1 \not\sim q_2$, because of the gadget of Lemma 13. Note that states with the same shape need not be k-similar.

Lemma 14. *For every subset $S \subseteq Q \setminus \{q_0\}$ of states and map* min-level: $S \to \mathbb{N}$, *there exists a DFA $N = \langle Q \cup I, \Sigma \cup \Delta, \mu, q_0, F\rangle$ such that $|\mu^{-1}(i)| = 1$ for every $i \in I$ and* $\text{in-level}_N(s) = \max(\text{in-level}_M(s), \text{min-level}(s))$ *for every $s \in S$.*

We will indicate the level i below the state name in graphical illustrations. Moreover, we add a special feathered arrow to the state q, whenever the gadget is used for the state q to increase its level.

We now present the construction. Let $G = \langle V, E\rangle$ be an undirected graph. Select $k, s \in \mathbb{N}$ such that $s > \log(|V|) + 2$ and $k > 4s$. Moreover, let $\ell = k - 2s$.

Definition 15. *We construct the DFA $M = \langle Q, \Sigma, \delta, 0, F\rangle$ as follows:*

- $Q = \{\bot, ☻, ☹, 3_0\} \cup \{i_j \mid i \in [2], j \in [\ell]\} \cup V \cup [0, s] \cup \{☺_i \mid 0 \le i \le s\}$,
- $\Sigma = \{a, b\} \cup V \cup E$,
- $F = \{☺_s, 1_\ell\}$, *and*
- *for every $v \in V$, $e = \{v_1, v_2\} \in E$ with $v \notin e$ and $v_1 < v_2$, $i \in [s]$, and $j \in [\ell]$*

$$\delta(i-1, a) = i \qquad \delta(v_1, e) = ☺_0 \qquad \delta(1_0, e) = ☻ \qquad \delta(1_{j-1}, b) = 1_j$$

$$\begin{array}{llll} \delta(i-1,b)=i & \delta(v_2,e)=\text{☹} & \delta(2_0,e)=\text{☻} & \delta(2_{j-1},b)=2_j \\ \delta(\text{☹}_{i-1},a)=\text{☹}_i & \delta(v,e)=\text{☻} & \delta(3_0,e)=\text{☻} & \delta(1_\ell,b)=\text{☹}_s \\ \delta(\text{☹}_{i-1},b)=\text{☹}_i & \delta(v,a)=1_1 & & \delta(2_\ell,b)=\text{☹}_s \\ \delta(\text{☻},a)=\text{☹}_1 & \delta(s,v)=v & & \end{array}$$

– *For all remaining cases, we set* $\delta(q,\sigma)=\bot$.

Finally, we show how to collapse k-similar states using a proper 3-colouring $c\colon V \to [3]$. We obtain the k-similar DFA $c(M)=\langle P,\Sigma,\mu,0,F\rangle$ from M by merging each state v into $c(v)_0$. In addition, for every edge $e=\{v_1,v_2\}\in E$, we let $\mu(c(v_1)_0,e)=\delta(v_1,e)$ and $\mu(c(v_2)_0,e)=\delta(v_2,e)$. Since the colouring c is proper, we have that $c(v_1)\neq c(v_2)$, which yields that μ is well-defined. For the remaining $i\in[3]\setminus\{c(v_1),c(v_2)\}$, we let $\mu(i_0,e)=\text{☹}_0$. All equivalent states (i.e., $\bot$ and ☹) are merged. The gadgets that were added to M survive and are added to $c(M)$. Naturally, if a certain state does no longer exist, then all transitions leading to or originating from it are deleted too. This applies for example to ☻.

Lemma 16. *There exists a k-minimal DFA N for M with at most*

$$2^{2s-1}\cdot|E|\cdot(|V|-2)+3\cdot 2^{s-1}\cdot|E|+2^{s+1}\cdot|V|$$

errors if and only if the input graph G is 3-colourable.

Corollary 17. *'Error-bounded k-minimisation' is NP-complete.*

References

1. Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. J. ACM 54(3) (2007)
2. Badr, A., Geffert, V., Shipman, I.: Hyper-minimizing minimized deterministic finite state automata. RAIRO Theoret. Inform. Appl. 43(1), 69–94 (2009)
3. Castiglione, G., Restivo, A., Sciortino, M.: Hopcroft's algorithm and cyclic automata. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 172–183. Springer, Heidelberg (2008)
4. 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)
5. Gries, D.: Describing an algorithm by Hopcroft. Acta Inf. 2(2), 97–109 (1973)
6. Holzer, M., Maletti, A.: An $n\log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. Theoret. Comput. Sci. 411(38–39), 3404–3413 (2010)
7. Hopcroft, J.E.: An $n\log n$ algorithm for minimizing states in a finite automaton. In: Kohavi, Z. (ed.) Theory of Machines and Computations, pp. 189–196. Academic Press, London (1971)
8. Maletti, A.: Better hyper-minimization— not as fast, but fewer errors. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 201–210. Springer, Heidelberg (2011)
9. Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms 51(2), 122–144 (2004)