Compact representation of finite automata with failure transitions

Henrik Björklund, Johanna Björklund, and Niklas Zechner

Department of Computing Science, Umeå University
90187 Umeå, Sweden
{henrikb, johanna, zechner}@cs.umu.se

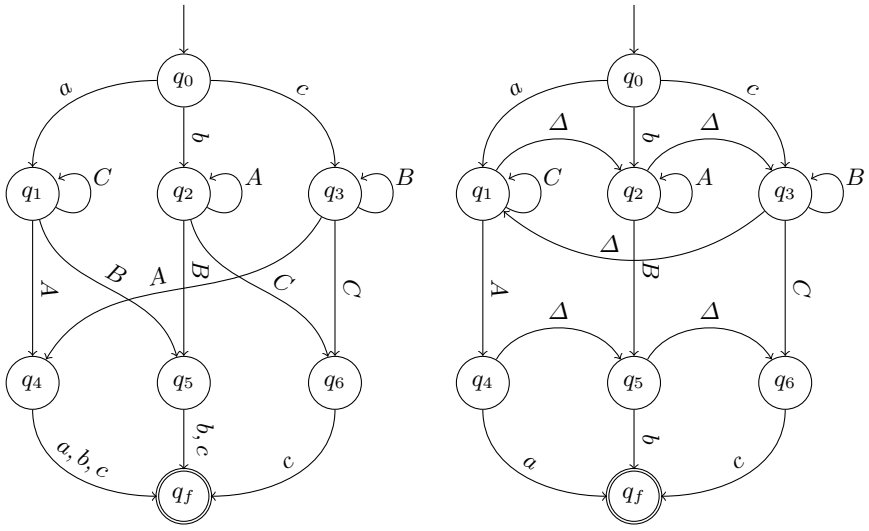
Abstract. Several linear-time algorithms for automata-based pattern matching rely on failure transitions for efficient back-tracking. Like epsilon transitions, failure transition do not consume input symbols, but unlike them, they may only be taken when no other transition is applicable. At a semantic level, this conveniently models catch-all clauses. Recent work demonstrates that failure transitions allow for compact representation of finite-automata in general. For devices with rich alphabets and dense transition functions in particular, the potential savings are great. In this paper, we prove that three important problems related to the introduction of failure transitions are NP-complete, but that efficient approximation is possible for at least one of them.

1 Introduction

Deterministic finite-state automata (DFA) offer a less compact representation than their non-deterministic counterpart, but despite this, they are often the preferred alternative in real-world applications. Their prevalence is largely due to convenient manipulation and to an efficiently solvable membership problem: Its time complexity in the uniform case is $O(|w| \cdot \log |Q|)$, where w is the input string and Q the state space. The corresponding figure for non-deterministic automata is $O(|w| \cdot |\delta|)$, where δ is the transition relation. A middle ground between compactness of representation and classification efficiency can be reached via *failure transitions*. Similar to epsilon transitions, these do not consume any input symbols, but unlike epsilon transitions, they can only be taken when there are no other applicable transitions.

When states in an automaton share a set of outgoing transitions, the automaton can be compressed by replacing these duplicates by a smaller number of failure transitions. Figure 1 (a) shows a state-minimal DFA over the alphabet $\{a, b, c\} \cup A \cup B \cup C$, where A, B, and C are the sets of symbols $\{a_i \mid i \in [k]\}$, $\{b_i \mid i \in [k]\}$, and $\{c_i \mid i \in [k]\}$, respectively, for some natural number k. A language-equivalent automaton in which regular transitions has been replaced by failure transitions is given in Figure 1 (b). The failure transitions are those labelled Δ. In this case, the failure transitions help save $3k-2$ transitions (more precisely, $3k+3$ regular transitions are saved and 5 failure transitions are added).

The addition of failure transitions does not preserve determinism in the classical sense of the word, but if the input automaton was deterministic and each state is allowed at most one outgoing failure transition, then the result is a *transition deterministic* automaton. In other words, although a deterministic automaton augmented with failure transitions can be in more states than one after reading part of the input string, every time it consumes an input symbol, the computation collapses onto a single state. As a consequence, the complexity of the membership problem only increases by a factor $|Q|$, where Q is the state set of the original DFA.



(a) A DFA (b) A language-equivalent FDFA

Fig. 1. A pair pf finite-state automata for the same language. The labels A, B, and C denote the sets of symbols $\{a_i \mid i \in [k]\}$, $\{b_i \mid i \in [k]\}$, and $\{c_i \mid i \in [k]\}$, respectively, for some natural number k.

Contributions

We prove that the Failure DFA (FDFA) transition-minimisation problem is NP-complete, thereby cancelling the search for an efficient and optimal algorithm initiated by [8]. We also show that a variant of the problem, where we are given a DFA and asked to save as many transitions as possible by introducing failure transitions, while preserving the basic structure of the DFA, is also NP-complete. In this case, however, we are able to provide an efficient approximation algorithm that saves at least 2/3 of the transitions an optimal algorithm could save. However, the question whether the transition minimisation problem can

also be approximated remains open. Finally, we solve a problem left open in [9] by showing that minimisation of binary automata is NP-complete.

Related work

Failure transitions make their first appearance in an article on pattern matching by Knuth et al. The authors give a linear-time algorithm for finding all occurrences of a pattern string within a text string. The algorithm reads the text string from left-to-right, while moving a pointer back and forth in the pattern string to remember what prefix of it has been encountered so far. Whenever the text string diverges from the pattern string, the algorithm backtracks by shifting the pointer according to a pre-computed failure function. [7]

Aho and Corasick build on this idea when they consider the problem of finding dictionary entries in an input string. The dictionary consists of a finite set of words L, and is represented as a prefix-tree acceptor A. Recall that this is a partial DFA recognising L, whose states are in one-to-one correspondence with the prefixes of L. To make A accept $\Sigma^* L \Sigma^*$, every state w is given a failure transition pointing to the longest suffix of w that is still a prefix of a string in $L\Sigma^*$. The advantage of failure transitions is that they save space, simplify the automata construction, and allow for efficient classification of input strings. [1]

Mohri, in turn, continues the work of Aho and Corasick, but takes as his starting point a DFA A recognising a possibly infinite set of target patterns. By traversing the states of A breadth-first, while adding failure transitions and auxiliary states, his algorithm produces a FDFA A' that recognises $\Sigma^* L$. The time complexity is linear in the size of A', which in the worst case is exponential in the size of A, but because of the failure transitions, the time complexity is not affected by the size of the alphabet. [10]

A survey of automata for pattern matching is given by [4]. In this context, failure transitions are sometimes treated under the name *suffix links* [12].

Recently, Kourie et al. considered the problem of using failure transitions to save as much space as possible. In other words, given an input DFA, try to find an equivalent automaton with failure transition whose total number of transitions is minimal. They develop two heuristic algorithms for solving the problem, but leave the complexity of the problem open. [8]

2 Preliminaries

For $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, \ldots, n\}$.

Automata. A *deterministic finite-state automaton* (abbreviated DFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, $\delta : Q \times \Sigma \to Q$ is a (potentially partial) *transition function*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is a set of *final states*. As usual, δ is generalised to $\delta' : Q \times \Sigma^* \to Q$ by letting $\delta'(q, \varepsilon) = q$ and $\delta'(q, aw) = \delta'(\delta(q, a), w)$ for every $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$.

A *failure DFA* (FDFA) is a tuple $B = (Q, \Sigma, \delta, \Delta, q_0, F)$ where $(Q, \Sigma, \delta, q_0, F)$ is a DFA and $\Delta : Q \to Q$ is a (potentially partial) *failure function*. Similarly to

the case for DFA, we derive from δ and Δ a partial function $\delta'' : Q \times \Sigma^* \to Q$. We still have $\delta''(q, \varepsilon) = q$, but now

$$\delta''(q, aw) = \begin{cases} \delta''(\delta(q, a), w) & \text{if } \delta(q, a) \text{ is defined, and} \\ \delta''(\Delta(q), aw) & \text{otherwise,} \end{cases}$$

for every $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$. Note that if Δ is not defined on q, or if its invocation causes infinite recursion, then the computation is aborted and the input string rejected.

From here on, we write δ also for δ' and δ''. The language accepted by a (failure) DFA A is then $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$.

Let $A = (Q, \Sigma, \delta, \Delta, F, q_0)$ be a (failure) DFA. Since we are concerned with reducing the number of transitions, we define the size of A as $|A| = |\delta|$ if A is a DFA, *or* $|A| = |\delta| + |\Delta|$ if it is a failure DFA. For $p \in Q$, we denote by Σ_p the set of symbols $\{a \in \Sigma \mid \exists q \in Q : \delta(p, a) = q\}$. The *abilities* of $p \in Q$ is the set $abil(p) = \{(a, q) \in \Sigma \times Q \mid \delta(p, a) = q\}$, and the *ability overlap* of the states $P \subseteq Q$ is the set $\bigcap_{p \in P} abil(p)$.

Problems of interest. The two main problems we study are the transition-minimisation problem and the failure-reduction problem.

Definition 1 (The transition-minimisation problem). *In the* transition-minimisation problem *we are given a DFA A and an integer k. The question is whether there is an FDFA B with at most k transitions such that $\mathcal{L}(B) = \mathcal{L}(A)$.*

Definition 2 (The failure-reduction problem). *The input to the* failure-reduction problem *is a DFA A and an integer k. The question is whether we can construct an FDFA B from A by substituting failure transitions for regular transitions, so that B has at least k* fewer *transitions than A and $\mathcal{L}(B) = \mathcal{L}(A)$.*

The main difference between the two problems are that in the transition-minimisation problem, we are not required to preserve any of the structure of the input DFA. In particular, we are allowed more states.

We will prove that both problems are, in general, NP-complete. It should be stressed that neither result follows immediately from the other. On the one hand, the freedom to add states could potentially make the problem easier. On the other hand, failure-reduction does not always produce a transition-minimal FDFA, which if it were the case could make that problem easier.

3 Basic properties of FDFAs

Before we address the subject matter, we make some basic observations that will be helpful later. The first of these is that FDFAs can be efficiently rewritten as language-equivalent DFAs by computing the closure of the failure transitions. The technique is quite similar to epsilon-removal.

Observation 3 Given an FDFA, we can construct an equivalent DFA with the same number of states in polynomial time.

Proof. Given an FDFA $B = (Q, \Sigma, \delta, \Delta, F, q_0)$ we construct an equivalent DFA $A = (Q, \Sigma, \delta', F, q_0)$. Notice that every part of A except for δ' is the same as the corresponding part of B. We change δ into δ' as follows. To begin with, we set $\delta' = \delta$. We then process the states in Q, possibly adding outgoing transitions. If $q_1 \in Q$ has no failure transition in B, the outgoing transitions from q_1 stay the same. If q_1 has a failure transition, let $q_1, q_2, \ldots q_k$ be the path of states reached by starting from q_1 and following Δ. In other words, $q_2 = \Delta(q_1)$, $q_3 = \Delta(\Delta(q_2))$, and so forth. If the path has a cycle, then q_k is the last state before the cycle closes. We look at the states on the path in order, starting with q_2. When we reach q_i, for every a such that q_1 does not yet have an outgoing transition labeled a in δ' and such that there is a p with $\delta(q_i, a) = p$, we let $\delta'(q_1, a) = p$. □

Observation 3 makes it clear that failure transitions may save on regular transitions, but not on states.

Observation 4 No FDFA for a language $\mathcal{L}$ can have fewer states than the state-minimal DFA for $\mathcal{L}$.

In fact, failure transitions are sometimes better leveraged by introducing *more* states. This situation is further discussed in the upcoming proof of Theorem 8.

Observation 5 For some languages $\mathcal{L}$, every transition-minimal FDFA for $\mathcal{L}$ has more states than the state-minimal DFA for $\mathcal{L}$.

By Observation 3, when given two FDFAs, we can construct equivalent DFAs, and then minimise and compare these, all in polynomial time.

Observation 6 Equivalence testing for FDFAs is polynomial.

However, unlike DFAs, FDFAs do not offer a canonical form of representation.

Observation 7 Given a language $\mathcal{L}$, there is, in general, no unique (up to homomorphism) state-minimal or transition-minimal FDFA for $\mathcal{L}$.

4 Transition minimisation

We first prove that the transition-minimisation problem is computationally hard. The proof is inspired by a proof by Jiang and Ravikumar, showing that the Normal Set Basis problem is NP-hard [6]. See also [2].

Theorem 8. *The transition-minimisation problem is NP-complete.*

Proof. The transition-minimisation problem is in NP since we can guess an FDFA with at most s transitions and test it for equivalence with the input DFA (viz. a FDFA without failure transitions) in polynomial time by Observation 6.

To show NP-hardness, we reduce from Vertex Cover. Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ and an integer k, we construct a DFA A_G

and an integer s such that there is a language-equivalent FDFA B_G that has at most s transitions if and only if G has a vertex cover of size at most k.

We first define the language $\mathcal{L}_G$ that A will accept. Let $V = \{v_1, \ldots, v_n\}$ and $\overline{E} = \{e_{i,j} \mid (v_i, v_j) \in E \ \wedge \ i < j\}$. We define the alphabet that $\mathcal{L}_G$ will use by $\Sigma = V \cup \overline{E} \cup \{a_i, b_i, c_i \mid v_i \in V\}$. Thus Σ has one symbol per vertex, one symbol per edge, and three extra symbols per vertex, so the size of Σ is $4n + m$.

The language $\mathcal{L}_G$ will only contain words of length two. The first symbol will be taken from $V \cup \overline{E}$ and the second symbol will depend on the first. To this end, we define the residual language of each member of $V \cup \overline{E}$ as follows.

$$res(v_i) = \{a_i, b_i, c_i\} \quad (\text{for } v_i \in V)$$
$$res(e_{i,j}) = \{b_i, c_i, b_j, c_j\} \quad (\text{for } e_{i,j} \in \overline{E})$$

We now define $\mathcal{L}_G$ by

$$\mathcal{L}_G = \left(\bigcup_{v_i \in V} v_i \cdot res(v_i) \right) \cup \left(\bigcup_{e_{i,j} \in \overline{E}} e_{i,j} \cdot res(e_{i,j}) \right).$$

The automaton A_G is simply the minimal DFA for $\mathcal{L}_G$; see the illustration in Figure 2. We note that A_G has $n + m + 2$ states and $4n + 5m$ transitions. The integer s will be $4n + 4m + k$.

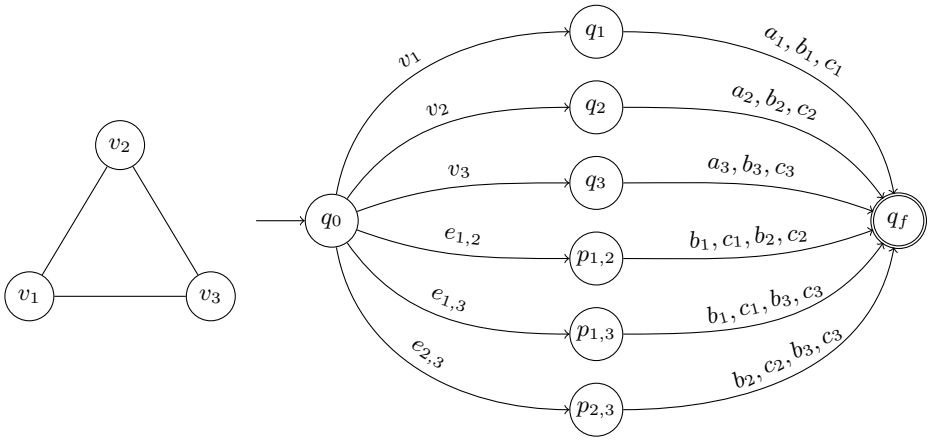


Fig. 2. A graph G and the corresponding DFA A_G

Let q_0 be the initial state of A_G and let q_f be the accepting state. For each $v_i \in V$, let q_i be the state A_G takes after reading v_i. Similarly, for each $e_{i,j} \in \overline{E}$, let $p_{i,j}$ be the state A_G takes after reading $e_{i,j}$.

Assume that G has a vertex cover of size k. We show how to construct B_G with s transitions such that $\mathcal{L}(B_G) = \mathcal{L}_G$. Let $C \subseteq V$ be a vertex cover for G of

size k. For every $v_i \in C$, do the following. Remove the transitions $\delta(q_i, b_i) = q_f$ and $\delta(q_i, c_i) = q_f$. Add a state r_i and the transitions $\Delta(q_i) = r_i$, $\delta(r_i, b_i) = q_f$, and $\delta(r_i, c_i) = q_f$. See Figure 3 for an illustration. The automaton now has $4n + 5m + k$ transitions, but we can save m transitions as follows.

For every $e_{i,j} \in \overline{E}$, we know that at least one of v_i and v_j belongs to C. Without loss of generality, assume that $v_i \in C$. We then remove the transitions $\delta(p_{i,j}, b_i) = q_f$ and $\delta(p_{i,j}, c_i) = q_f$ and add the failure transition $\Delta(p_{i,j}) = r_i$. This saves one transition. Since we can do this for every edge, we save m transitions and arrive at an automaton with $s = 4n + 4m + k$ transitions.

For the other direction, assume that there is an FDFA $B_G = (Q, \delta, \Delta, F, q_0)$ for $\mathcal{L}_G$ with s transitions. We argue that G must have a vertex cover of size k.

First, since all words in $\mathcal{L}_G$ have length two, Q contains three disjoint sets: those reachable after reading 0, 1, or 2 symbols, respectively. The first set is the singleton $\{q_0\}$. The third set can also be assumed to be a singleton $\{q_f\} = F$. As for the middle set, it has to have at least $n + m$ states, one for each possible first symbol. The reason for this is that all the symbols in $V \cup \overline{E}$ have different residual languages. Let $Q_1 = \{q_i \mid v_i \in V\} \cup \{p_{i,j} \mid e_{i,j} \in \overline{E}\}$ be the states reached by reading one symbol (before taking any failure transitions).

We also notice that no state in Q_1 can have a failure transition to another state in Q_1, since for every pair $t_i, t_j \in Q_1$, neither $\Sigma_{t_i} \subseteq \Sigma_{t_j}$ nor $\Sigma_{t_j} \subseteq \Sigma_{t_i}$. This means that every failure transition must lead to a state that is not in Q_1.

Creating new states and failure transitions can only save transitions when states in Q_1 have overlapping residual languages. The only case where this happens is when every "edge state" $q_{i,j}$ has overlapping residual languages with q_i and q_j. In the case of q_i the overlap is $\{b_i, c_i\}$ and in the case of q_j it is $\{b_j, c_j\}$.

It follows that the only way failure edges can save transitions is to let states q_i fail to a new state r_i on b_i and c_i, let r_i lead to q_f on b_i and c_i and let states $p_{i,j}$ or $p_{j,i}$ also fail to r_i on b_i and c_i. We can count the savings we achieve in the following way. For every q_i we add a failure edge to, we get one extra transition. For every $p_{i,j}$, on the other hand, that can fail to an r_i corresponding to an incident vertex, we save one transition.

If B_G has $s = 4n+4m+k$ transitions, this means that we have "saved" $m-k$ transitions. Assume that we have added failure edges to k' "vertex states" q_i. How many "edge states" must then have received failure edges? Let this number be ℓ. We get $\ell - k' = m - k$. Notice that we must have $k' \leq k$, since $\ell \leq m$. If $k' = k$, then $\ell = m$ and we immediately have that G has a vertex cover of size k. If, on the other hand, $k' < k$, we note that $m - \ell = k - k'$. In other words, the number of edges that are not using failure transitions equals k minus the number of vertices that are using failure transitions. We can now construct a vertex cover for G as follows. Include the k' vertices whose corresponding states in B_G have failure transitions in the cover. This leaves $k - k'$ edges uncovered. For each such edge, we select one of its endpoints arbitrarily and include it in the cover. The result is a cover of size k for all the edges. □

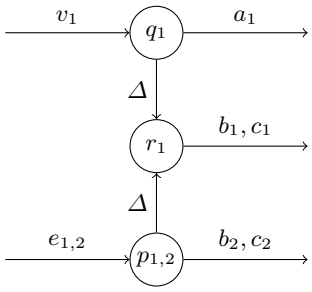


Fig. 3. The vertex state q_1 has been extended with failure state r_1 and the edge state $p_{1,2}$ has attached to it.

5 Failure reduction

In this section, we study the failure-reduction problem. We first establish that it is NP-complete and then give an efficient algorithm which approximates it within a factor 2/3.

Theorem 9. *The failure-reduction problem is NP-complete.*

Proof. The problem is in NP since, by Observation 6, equivalence testing for FDFAs is polynomial. Given a DFA A and an integer k, we can guess an FDFA with k fewer transitions than A and verify that it is equivalent to A.

For NP-hardness, we reduce from Hamiltonian Cycle. Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ we construct a DFA $A = (Q, \Sigma, \delta, I, F)$ such that there is an FDFA B for $\mathcal{L}(A)$ with $k = n(n-2)$ fewer transitions if and only if G has a Hamiltonian cycle.

Let $V = \{v_1, \ldots, v_n\}$. The alphabet Σ contains a letter for each vertex and for each edge of G, i.e., $\Sigma = V \cup \{e_{i,j} \mid i < j \wedge (v_i, v_j) \in E\}$. The state set of A is $Q = \{q_I, q_F\} \cup \{p_1, \ldots, p_n\}$, with $I = \{q_I\}$ and $F = \{q_F\}$. We now describe the transition function of A in detail.

- For every vertex name v_i, $\delta(q_I, v_i) = p_i$.
- Every state $p_i \in \{p_1, \ldots, p_n\}$ has the following outgoing transitions.
  - $\delta(p_i, v_i) = p_i$,
  - $\delta(p_i, v_j) = q_F$ for every $v_j \neq v_i$,
  - $\delta(p_i, e_{j,\ell}) = q_F$ for every edge name $e_{j,\ell}$ such that $i = j$ or $i = \ell$,
  - $\delta(p_i, e_{j,\ell}) = p_i$ for every edge name $e_{j,\ell}$ such that $i \neq j$ and $i \neq \ell$.

This means that the language $\mathcal{L}(A)$ of A conisists of all words $v_i \tau_i^* \sigma_i$, where τ_i contains v_i and the names of all edges that are *not* adjacent to v_i, while σ_i contains $V \setminus \{v_i\}$ and the names of all edges that *are* adjacent to v_i. Let $\mathcal{L}_G = \mathcal{L}(A)$. It is straigtforward to verify that A is the minimal DFA for $\mathcal{L}_G$. Notice that q_I has n outgoing transitions, q_F has none and q_i has $n + m$, for every $i \in [n]$. In total, A has $n + n(n + m) = n(n + m + 1)$ transitions.

First, we assume that G has a Hamiltonian cycle and show that there is an FDFA B with $k = n(n-2)$ fewer transitions than A such that $\mathcal{L}(B) = \mathcal{L}_G$. By renaming vertices, we can assume that the cycle is $v_1 \to v_2 \to \cdots \to v_n \to v_1$. We construct B from A by adding a failure transition $\Delta(p_i) = p_{i+1}$ for every $i \in [n-1]$ and the failure transition $\Delta(p_n) = p_1$. All transitions that have been made redundant are then removed. After this, q_I still has n outgoing transitions, while q_F has none. We argue that every p_i, for $i \in [n]$ has $m+2$ outgoing transitions, i.e., $n-2$ fewer than in A. Indeed, looking at p_i and p_{i+1} (or p_1, if $i = n$), we see that in A, they both have transitions to q_F for every v_j such that $j \notin \{i, i+1\}$. Thus $n-2$ transitions can be removed from p_i. Additionally they both have transitions to q_F on the edge name $e_{i,i+1}$. Thus we can remove one additional outgoing transition from p_i. On the other hand, we have added a failure transition from p_i. This means that in total, p_i has $n-2$ outgoing transitions fewer in B than in A. This means that B has $n(n-2)$ fewer transitions than A, as required.

Next, we assume that there is an FDFA $B = (Q, \Sigma, \delta', \Delta, I, F)$ with k transitions fewer than A and such that $\mathcal{L}(B) = \mathcal{L}_G$ and argue that G must have a Hamiltonian cycle.

There has to be n transitions leaving q_I, one for each vertex name v_i. We can assume that these are the transitions $\delta(q_I, v_i) = p_i$. On the other hand, no transitions need to leave q_F. Thus we can focus on the transitions from the states $p_1, \ldots, p_n$. Each failure transition will go from one such state to another such state. No pair of such states can share more than $n-1$ abilities, which means that each such state will have *at least* $m + n - (n-1) + 1 = m + 2$ outgoing transitions. This means that B will have *at most* $k = n(n-2)$ transitions fewer than A and that, for this number to be realised, each state in $\{p_1, \ldots, p_n\}$ must have exactly $m+2$ outgoing transitions.

In A, each such state has one transition per edge name and one per state name, i.e., $n+m$ outgoing transitions. Thus every such state in B must have a failure transition. Assume that there is a failure transition from p_i to p_j. Then we can remove the $n-2$ outgoing transitions on the vertex names $V \setminus \{v_i, v_j\}$ from p_i. On the other hand, we have added a failure transition, leaving us with $m+3$ transitions. This means that for p_i to have only $m+2$ transitions, it has to share one more ability with p_j. This is only possible if there is an edge between v_i and v_j in G. In this case, both states have transitions to p_F on $e_{i,j}$.

Next, we argue that the graph of the failure function Δ must be connected and cyclic. Note that if there is a failure transition from p_i to p_j, then p_i must have a transition to itself on v_i and to q_F on v_j. These are its only transitions on vertex names. This also means that for all transitions on vertex names to q_F to be represented somewhere, there can be no two states that fail to the same state. Since each such transition must be reachable via failure transitions from all but one state in $\{p_1, \ldots, p_n\}$, the graph of Δ is indeed connected and cyclic.

As shown above, each edge of the graph of Δ also corresponds to an edge in G. Thus Δ induces a Hamiltonian cycle on G. □

Since the transition reduction problem is computationally hard, approximate solutions may be valuable. We next suggest a fast and easily implemented algorithm that saves at least two-thirds as many transitions as an optimal algorithm.

Throughout the following, $A = (Q, \Sigma, \delta, I, F)$ is a DFA, and our objective is to construct a language-equivalent FDFA with as few transitions as possible by adding failure transitions and removing regular transitions. To guide the placement of failure transitions, we define the *prospect graph* for A. Intuitively, the graph tells us between what states failure transitions are useful and allowable: We only want to add failure transitions if they save us regular transitions, and we should not add transitions that alter the accepted language.

Definition 10 (Prospect graph). *The* prospect graph *for A is the weighted directed graph $P(A) = (Q, E, w)$, with*

$$E = \{(p, q) \mid \mathit{abil}(p) \cap \mathit{abil}(q) \neq \emptyset \text{ and } \Sigma_p \subseteq \Sigma_q\} \ ,$$

and $w((p, q)) = |\mathit{abil}(p) \cap \mathit{abil}(q)| - 1$, for every $(p, q) \in E$.

The prospect graph for the DFA of Figure 1 is shown in Figure 4. By adding a failure transition between any two of the states q_1, q_2, and q_k, we can save k regular transitions at the cost of one failure transition. We may also add a failure transition from q_4 to q_5 or q_6, thereby saving 0 or 1 transitions, but the opposite direction is not allowed: if a failure transition were added from q_6 to q_4 it would be possible to read the symbol a from q_6, and this would increase the language.

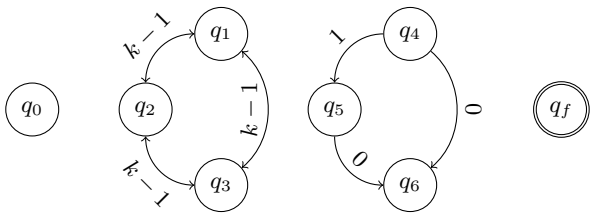


Fig. 4. The prospect graph for the DFA in Figure 1 (a)

Lemma 11. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA and $B = (Q, \Sigma, \delta_B, \Delta_B, q_0, F)$ a transition-minimal language-equivalent FDFA that can be constructed from A by adding failure-transitions and removing redundant regular transitions. Let $k = |A| - |B|$. There is a language-equivalent FDFA $C = (Q, \Sigma, \delta_C, \Delta_C, q_0, F)$ such that $k' = |A| - |C| \geq 2k/3$ and such that Δ_C is acyclic.*

Proof. We first show that every cycle in Δ_B is of length 3 or more. Suppose that B has a failure cycle of length two through states p and q. We know then that $\Sigma_p = \Sigma_q$, since the states can fail to each other. By removing the failure transition from p to q, and moving all transitions on tuples in $\mathit{abil}(q) \cap \mathit{abil}(p)$

from q to p, we obtain a smaller automaton. Since the operation preserves the residual languages of p and q, the new automaton is language-equivalent with the original one, contrary to the minimality assumption.

By repeatedly removing from each cycle of Δ_B the failure transition that saves the least regular transitions, the failure function can be made acyclic. Since Δ_B has out-degree at most 1, no edge can belong to more than one cycle. According to the reasoning above, it suffices to drop at most one third of the edges to clear all cycles. For each failure edge that is removed, at least two will remain, and each of them will save at least as many regular transitions as the removed edge. This means that when all cycles have been eliminated, we are left with a failure function Δ_C that saves at least 2/3 as many transitions as Δ_B. □

Each failure function Δ on Q describes a function graph (Q, Δ), i.e., a graph where each node has out-degree at most one.

Observation 12 Let $G = (V, E, w)$ be a directed graph with positive edge weights. Let $\Delta \subseteq E$ be such that (V, Δ) is an acyclic function graph. Then (V, Δ^{-1}) is a *branching*, i.e., an acyclic directed graph such that no vertex has in-degree larger than 1. Further more, if (V, Δ^{-1}, w) is a *maximum weight* branching on (V, E^{-1}, w), then (V, Δ, w) is a maximum weight acyclic function graph on $G = (V, E, w)$.

Theorem 13 below now follows immediately from the fact that it is possible to find a maximum branching on the prospect graph in polynomial time. An algorithm for this problem was discovered in the 1960s by Chu and Liu [3] and, independently, by Edmonds [5]. An implementation that runs in time $O(|E| \log |V|)$ was later suggested by Tarjan [11].

Theorem 13. *The failure-reduction problem can be approximated within a factor* 2/3 *in polynomial time.*

The automaton in Figure 1 (b) is a transition-minimal state-minimal FDFA for $L(A)$. Since its failure function contains a cycle, the above approximation technique will not find it, but it will find the FDFA in Figure 5 which saves $2k - 1$ transitions.

6 Minimisation of binary automata

Binary automata (BFDFAs) are a restricted form of FDFAs, introduced by Kowaltowski, Lucchesi and Stolfi in [9]. An FDFA $B = (Q, \Sigma, \delta, \Delta, q_0, F)$ is a BFDFA if there is at most one non-failure transition from each state, i.e, for every $p \in Q$ there is at most one $a \in \Sigma$ such that $\delta(p, a)$ is defined. This means that the automaton can be represented as a set of four-tuples (p, a, q, q'), with $\delta(p, a) = q$ and $\Delta(p) = q'$. To minimise a BFDFA means to minimise the number of such tuples. It was conjectured in [9] that this problem is NP-complete. We show that this is indeed the case. To save space, the proof of the following theorem has been moved to Appendix A.

Theorem 14. *The minimisation problem for binary automata is NP-complete.*

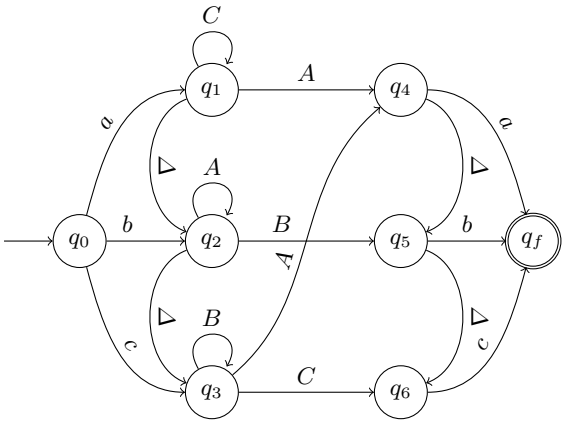


Fig. 5. An FDFA with acyclic failure function that accepts the same language as the DFA in Figure 1 (a)

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.
2. H. Björklund and W. Martens. The tractability frontier for NFA minimization. *Journal of Computer and System Sciences*, 78(1):198–210, 2012.
3. Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.
4. M. Crouchemore and C. Hancart. Automata for matching patterns. In *Handbook of Formal Languages*, volume 2, pages 399–462. Springer, 1997.
5. J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.
6. T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal of Computing*, 22(6):1117–1141, 1993.
7. D. E. Knuth, J. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.
8. D. G. Kourie, B. W. Watson, L. G. Cleophas, and F. Venter. Failure deterministic finite automata. In *Proceedings of the Prague Stringology Conference 2012, Prague, Czech Republic*, pages 28–41, 2012.
9. T. Kowaltowski, L. L. Lucchesi, and J. Stolfi. Minimization of binary automata. In *Proceedings of the First South American String Processing Workshop, Belo Horizonte, Brasil*, pages 105–116, 1993.
10. M. Mohri. String-matching with automata. *Nordic Journal of Computing*, 4(2):217–231, 1997.
11. R. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.
12. P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA*, pages 1–11, 1973.

A Binary automata

Proof (of Theorem 14). For membership, it is enough to notice that for every BFDFA B, just as for every FDFA, an equivalent DFA A_B can be constructed in polynomial time. Thus a nondeterministic algorithm can, given B, guess a sufficiently small BFDFA B', construct A_B and $A_{B'}$, minimise them, and check for equivalence.

For NP-hardness, we again reduce from Vertex Cover. Given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ and an integer k, we will construct a BFDFA B_G and an integer s such that the minimal BFDFA for $\mathcal{L}(B_G)$ has s or fewer tuples if and only if G has a vertex cover of size k.

We first define $\mathcal{L}(B_G)$. Let $V = \{v_1, \ldots, v_n\}$. We will use names for the vertices and edges of G as letters for our alphabet Σ. Thus

$$\Sigma = V \cup \{e_{i,j} \mid i < j \ \wedge \ (v_i, v_j) \in E\}.$$

We now define our language by

$$\mathcal{L}(B_G) = \bigcup_{(v_i,v_j)\in E} (e_{i,j} \cdot (v_i + v_j)) .$$

In other words, $\mathcal{L}(B_G)$ contains edge names followed by the name of one of the vertices incident to the edge. In particular, all strings in $\mathcal{L}(B_G)$ have length two and the language is thus finite.

Given $\mathcal{L}(B_G)$ we can trivially construct B_G with $3m$ tuples. What we will show is that there is an equivalent BFDFA B'_G with $s = 2m + k + 1$ tuples if and only if G has a vertex cover of size k.

Assume that $C \subseteq V$ is a vertex cover for G and that $|C| = k$. We construct $B'_G = (Q, \delta, F, \Delta, q_0)$ as follows. For every edge (v_i, v_j) in E, there are two states, $p_{i,j}$ and $q_{i,j}$ in Q. Additionally, Q has one state r_i for every vertex v_i in the cover C. Finally, Q has an accepting state $\top$ and a rejecting state $\bot$. In total,

$$Q = \{p_{i,j}, q_{i,j} \mid i < j \ \wedge \ (v_i, v_j) \in E\} \cup \{r_i \mid v_i \in C\} \cup \{\top, \bot\}.$$

Let $\prec$ be the lexicographical ordering on the edge names $e_{i,j}$, i.e., $e_{i,j} \prec e_{i',j'}$ if $i < i'$ or if $i = i'$ and $j < j'$. We will also use this ordering on the corresponding sets of states. For a state $p_{i,j}$ we write $\mathrm{Next}(p_{i,j})$ for the state that comes next in this ordering. The initial state of B'_G is $q_0 = \min_{\prec}\{p_{i,j}\}$. For every edge name $e_{i,j}$, we set $\delta(p_{i,j}, e_{i,j}) = q_{i,j}$. For every edge name $e_{i,j}$ except $e_{t,t'} = \max_{\prec}\{e_{i,j}\}$ we also set $\Delta(p_{i,j}) = \mathrm{Next}(p_{i,j})$. For $e_{t,t'}$ we set $\delta(p_{t,t'}) = \bot$. Next, we describe the transitions leaving the states $q_{i,j}$. By assumption, either v_i or v_j (or both) belongs to C. Assume, without loss of generality, that $v_i \in C$. Then we set $\delta(q_{i,j}, v_j) = \top$ and $\Delta(q_{i,j}) = r_j$. For the states r_i, we set $\delta(r_i, v_i) = \top$ and $\Delta(r_i) = \bot$. Finally, we set $\Delta(\top) = \bot$. This completes the description of B'_G. If we represent it as four-tuples, it will have one tuple per state, except for $\bot$. Thus it has $2m + k + 1$ tuples. It should be clear that B'_G accepts $\mathcal{L}(B_G)$.

We now need to show that if G has no vertex cover of size k, then there is no BFDFA for $\mathcal{L}(B_G)$ with s or fewer tuples. Since each state can have only one

transition that reads a letter, there must be m four-tuples where the letter is an edge name. We can now ask how many different states we can be in after having just read one letter and not taken any failure transitions after that. Notice that for each edge name, the residual language is unique. In other words, there are no two edge names $e_{i,j}$ and $e_{i',j'}$ such that the sets of suffixes we can read after them to complete a string in $\mathcal{L}(B_G)$ are identical. Thus there must be m different states that we can be in directly after reading an edge name. Each such state contributes another tuple. These cannot, however, be the only states from which we can read a vertex name. Indeed, from each such state, we should be able to read two distinct vertex names. Thus there must be some extra states, which these states can fail to, and from where we can read exactly one vertex name. If two edge names represent edges that share an incident vertex, then the corresponding states could share an extra state. Therefore the smallest number of extra states is equal to the size of the smallest set of vertices such that each edge has at least one incident vertex in the set, or, in other words, the size of the smallest vertex cover for G. Additionally, we will need an accepting state and its corresponding tuple. Thus, if G has no vertex cover of size k, then there can be no BFDFA for $\mathcal{L}(B_G)$ of size smaller than $2m + k + 1$. □