

Stretching and Jamming of Automata

NOUD DE BEIJER

Department of Mathematics and Computing Science
Eindhoven University of Technology, The Netherlands
and

BRUCE W. WATSON AND DERRICK G. KOURIE

FASTAR Research Group
Department of Computer Science
University of Pretoria, South Africa

In this paper we present two transformations on automata, called stretching and jamming. These transformations will, under certain conditions, reduce the size of the transition table, and under other conditions reduce the string recognition time. Given a deterministic finite automaton, we can stretch it by transforming each single transition into two or more sequential transitions, thereby introducing additional intermediate states. Jamming is the opposite transformation, in which two or more successive transitions are transformed into a single transition, thereby removing redundant intermediate states.

We will present formal definitions of stretching and jamming. We will give algorithms for stretching and jamming and we will calculate theoretical bounds, when stretching/jamming is effective in terms of memory consumption and string recognition time.

Categories and Subject Descriptors: F.1.1 [COMPUTATION BY ABSTRACT DEVICES]: Models of Computation—Automata; C.4 [PERFORMANCE OF SYSTEMS]: Modeling techniques; I.5.1 [PATTERN RECOGNITION]: Models—Structural

General Terms: Algorithms, Languages, Performance, Theory

Additional Key Words and Phrases: stretching, jamming, transformation, performance, transition table, string recognition time

1. INTRODUCTION

In a compiler, a lexical analyzer is used to read input characters and to produce as output a sequence of tokens that the parser uses for syntax analysis [Aho et al. 1986, p. 84]. Since the process of lexical analysis occupies a reasonable portion of the compiler's time, the lexical analyzer should minimize the number of operations it performs per input character [Aho et al. 1986, p. 144].

The lexical analyzer uses finite automata to recognize languages. There are different ways to implement the transition function of a finite automaton. The easiest and fastest way is to use a transition table in which there is a row for each state and a column for each input symbol. Unfortunately, this representation can take up a lot of space [Aho et al. 1986, p. 114].

Thus, transformations on automata that increase their performance in terms of memory consumption or string recognition time are potentially useful (see for example [Watson 1995]).

We propose two transformations on automata and regular expressions: *stretching* and *jamming*. Under certain conditions, these transformations will produce more efficient automata in terms of memory consumption and string recognition time.

Given a deterministic finite automaton (DFA), we can stretch it by transforming each single transition into two or more sequential transitions, thereby introducing additional intermediate states. For example, an ASCII DFA can be stretched by transforming each single ASCII (8-bit) character transition into two transitions, each of 4-bits.

Jamming is the opposite transformation, in which two successive transitions (based on, for example, input

Author Addresses:

N de Beijer, Department of Mathematics and Computing Science, Eindhoven University of Technology, Den Dolech 2, Postbus 513, 5600 MB Eindhoven, The Netherlands;a.a.d.beijer@stud.tue.nl

B W Watson and D G Kourie, FASTAR Research Group (<http://fastar.cs.up.ac.za/>), Department of Computer Science, University of Pretoria, Pretoria 0002, South Africa;bruce@bruce-watson.com;dkourie@cs.up.ac.za

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, that the copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than SAICSIT or the ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2003 SAICSIT

characters represented in 8-bits) are transformed into a single transition (in this example based on an input character represented in 16-bits) .

In the next section we first give the basic notions and notations that are used in this paper. In section 3, we present the definitions of stretching and jamming. Next, we present an application of stretching and jamming called bit-level stretching and jamming followed by the algorithms. In the last section we present our conclusions.

2. PRELIMINARIES

In this section we present the basic notions and notations used in this paper. Most of the notations used are standard, (see for example [Hopcroft et al. 2001]), but a few new notations are introduced.

A deterministic finite automaton, *DFA*, is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is the *alphabet*, $\delta : Q \times \Sigma \rightarrow Q$ is the (partial) *transition function*, q_0 is the *initial state* and F is a subset of Q whose elements are *final states*. $|Q|$ is the number of states and $|\Sigma|$ is the number of elements in the alphabet, or *alphabet size*. Σ^+ is the *plus closure* of the alphabet, the set of strings obtained by concatenating one or more symbols from Σ . $|Q||\Sigma|$ is the *transition table size*.

Note that since cells represent states, the minimum cell size is determined by the minimum space requirements to represent a state, which is in turn determined by the total number of states. Although stretching and jamming will change the number of states in an automaton for the present we will assume that the transition table cell size does not change in either transformation. We expect that in most cases the practical effects of this assumption are unlikely to be significant, but this has to be examined in practice.

A transition in a DFA M from p to q with label a will be denoted by (p, a, q) where $(p, a, q) \in Q \times \Sigma \times Q$ and $q = \delta(p, a)$.

A *path* of length k in a DFA M is a sequence $\langle (r_0, a_0, r_1), \dots, (r_{k-1}, a_{k-1}, r_k) \rangle$, where $(r_i, a_i, r_{i+1}) \in Q \times \Sigma \times Q$ and $r_{i+1} = \delta(r_i, a_i)$ for $0 \leq i < k$. The word $a_0 a_1 \dots a_{k-1} \in \Sigma^k$ is the *label* of the path.

The *extended transition function* of a DFA M , $\hat{\delta} : Q \times \Sigma^+ \rightarrow Q$, is defined so that $\hat{\delta}(r_i, w) = r_j$ iff there is a path from r_i to r_j , labeled w .

A nondeterministic finite automaton, *NFA*, is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, defined in the same way as a DFA, with the following exception: $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is the transition function. Note that $\mathcal{P}(Q)$ is the powerset of Q , the set consisting of all possible subsets of Q , this is sometimes written as 2^Q in the literature.

A transition in an NFA M from p to q with label a will also be denoted by (p, a, q) where $(p, a, q) \in Q \times \Sigma \times Q$ and $q \in \delta(p, a)$.

A path of length k in an NFA M is a sequence $\langle (r_0, a_0, r_1), \dots, (r_{k-1}, a_{k-1}, r_k) \rangle$, where $(r_i, a_i, r_{i+1}) \in Q \times \Sigma \times Q$ and $r_{i+1} \in \delta(r_i, a_i)$ for $0 \leq i < k$.

In an NFA M , the extended transition function, $\hat{\delta} : Q \times \Sigma^+ \rightarrow \mathcal{P}(Q)$, is also defined so that $r_j \in \hat{\delta}(r_i, w)$ iff there is a path from r_i to r_j , labeled w .

If there is a path of length k from r_0 to r_k with label w in a DFA or an NFA we will denote this by $[r_0, w, r_k]$.

A state with one in-going and one out-going transition will be called an *unbranched state*. A path, consisting only of unbranched states (with the possible exception of the start- and/or end state) is an *unbranched path*. If path $[r_0, w, r_k]$ is an unbranched path it will be denoted by $\langle\langle r_0, w, r_k \rangle\rangle$.

3. DEFINITIONS OF STRETCHING AND JAMMING

In this section we give formal definitions of stretching and jamming. One way in which we can stretch a DFA is by transforming each transition into k sequential transitions, and $k - 1$ additional intermediate states. Note that for simplicity we only regard stretching DFAs, not NFAs. In future publications we will present the stretching of NFAs. In some cases, the automaton resulting from stretching may have more than one transition with the same label from a given state and therefore the result of stretching will be an NFA.

If DFA FA_0 can be stretched into NFA FA_1 , we call FA_1 a stretch of FA_0 . The set of states of FA_1 consists of a subset S_1 of ‘normal’ states and a subset I of additional intermediate states. These additional intermediate states are unbranched states. Furthermore, there is an injection from the alphabet of FA_0 , Σ_0 to Σ_1^+ , the plus closure of the alphabet of FA_1 , and for every transition in FA_0 there are two or more sequential transitions in FA_1 .

Definition 3.1. Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be a DFA, and let $FA_1 = (S_1 \cup I, \Sigma_1, \delta_1, q_1, F_1)$ be an NFA. FA_1 is a stretch of FA_0 iff:

—There is an injection $\tau : \Sigma_0 \rightarrow \Sigma_1^+$, thus:

$$(\forall a \in \Sigma_0 : (\exists w \in \Sigma_1^+ : \tau(a) = w))$$

—There is a bijection $\varphi : S_0 \leftrightarrow S_1$, with the following properties:

$$\begin{aligned} &-\varphi(q_0) = q_1 \\ &-(\forall f_0 \in F_0 : (\exists f_1 \in F_1 : \varphi(f_0) = f_1)) \wedge (\forall f_1 \in F_1 : (\exists f_0 \in F_0 : \varphi(f_0) = f_1)) \\ &-(\forall (p, a, q) : (\exists k, \langle \langle r_0, w, r_k \rangle \rangle : \varphi(p) = r_0 \wedge \varphi(q) = r_k \wedge \tau(a) = w)), \text{ where } r_{i+1} \in \delta_1(r_i, a_i), r_0, r_k \in S_1, \\ &\quad r_1, \dots, r_{k-1} \in I \text{ and } w = a_0 \cdots a_{k-1}, \text{ for } 0 \leq i < k. \end{aligned}$$

We define jamming as the opposite transformation of stretching. Because of symmetry we will consider only the jamming of NFAs into DFAs. Again, in future work we intend to consider the jamming of DFAs.

If NFA FA_0 is jammed into DFA FA_1 (FA_1 is a jam of FA_0) then FA_0 is a stretch of FA_1 . The set of states of FA_0 consists of a subset S_0 of ‘normal’ states and a subset R of redundant intermediate states. These redundant intermediate states are unbranched states, and will be removed by the jamming transformation.

Definition 3.2. Let $FA_0 = (S_0 \cup R, \Sigma_0, \delta_0, q_0, F_0)$ be an NFA, and let $FA_1 = (S_1, \Sigma_1, \delta_1, q_1, F_1)$ be a DFA. FA_1 is a jam of FA_0 iff FA_0 is a stretch of FA_1 , with R being the set of additional intermediate states, resulting from stretching.

We now introduce the factor f in stretching and jamming. If DFA FA_0 is stretched by a factor f into NFA FA_1 (FA_1 is an ‘f-stretch’ of FA_0) then there is an injection between Σ_0 and Σ_1^f . Furthermore, for each transition in FA_0 there is exactly f sequential transitions in FA_1 .

Definition 3.3. Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be a DFA, and let $FA_1 = (S_1 \cup I, \Sigma_1, \delta_1, q_1, F_1)$ be an NFA. FA_1 is an ‘f-stretch’ of FA_0 iff:

— FA_1 is a stretch of FA_0 , such that the injection τ specializes to an injection $\tau : \Sigma_0 \leftrightarrow \Sigma_1^f$, thus:

$$(\forall a \in \Sigma_0 : (\exists w \in \Sigma_1^f : \tau(a) = w))$$

—The bijection $\varphi : S_0 \leftrightarrow S_1$, is characterized by:

$$\begin{aligned} &-\varphi(q_0) = q_1 \\ &-(\forall f_0 \in F_0 : (\exists f_1 \in F_1 : \varphi(f_0) = f_1)) \wedge (\forall f_1 \in F_1 : (\exists f_0 \in F_0 : \varphi(f_0) = f_1)) \\ &-(\forall (p, a, q) : (\exists \langle \langle r_0, w, r_f \rangle \rangle : \varphi(p) = r_0 \wedge \varphi(q) = r_f \wedge \tau(a) = w)), \text{ where } r_{i+1} \in \delta_1(r_i, a_i), r_0, r_f \in S_1, \\ &\quad r_1, \dots, r_{f-1} \in I \text{ and } w = a_0 \cdots a_{f-1}, \text{ for } 0 \leq i < f. \end{aligned}$$

Jamming by a factor f is defined analogously to stretching by a factor f . Note that, by definition, jamming by a factor f is not always possible. NFA FA_0 can only be jammed if there exists a DFA FA_1 which can be stretched into FA_0 . For example, an NFA which has no unbranched states cannot be jammed.

Definition 3.4. Let $FA_0 = (S_0 \cup R, \Sigma_0, \delta_0, q_0, F_0)$ be an NFA, and let $FA_1 = (S_1, \Sigma_1, \delta_1, q_1, F_1)$ be a DFA. FA_1 is an ‘f-jam’ of FA_0 iff FA_0 is an ‘f-stretch’ of FA_1 .

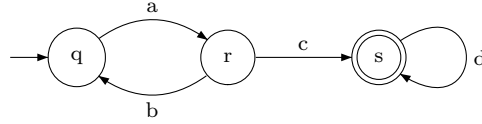


Figure 1. DFA FA_0

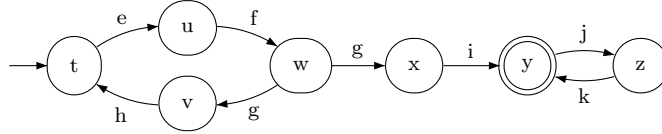


Figure 2. NFA FA_1 , a 2-stretch of DFA FA_0

$$\begin{aligned} \tau(a) &= ef \\ \tau(b) &= gh \\ \tau(c) &= gi \\ \tau(d) &= jk \end{aligned}$$

Figure 3. Injection τ

$$\begin{aligned} \varphi(q) &= t \\ \varphi(r) &= w \\ \varphi(s) &= y \end{aligned}$$

Figure 4. Bijection φ

EXAMPLE 3.5. To illustrate these definitions we give an example. The graph in figure 1 represents the DFA $FA_0 = (\{q, r, s\}, \{a, b, c, d\}, \delta_0, q, \{s\})$. DFA FA_0 can be stretched by a factor 2 into NFA FA_1 of figure 2. NFA $FA_1 = (S_1 \cup I, \{e, f, g, h, i, j, k\}, \delta_1, t, \{y\})$, with $S_1 = \{t, w, y\}$ and I , the set of additional intermediate states, is $\{u, v, x, z\}$. Injection τ and bijection φ are shown in figure 3 and 4 respectively.

4. BIT-LEVEL STRETCHING AND JAMMING

4.1 Introduction

In this section we present an application of stretching and jamming. We look at stretching and jamming on a bit-level. We will only consider automata in which each element of the alphabet is a bit string. An n -bit automaton is an automaton whose alphabet consists of all the 2^n bit strings of length n .

PROPOSITION 4.1. Let f be a factor of n . Then we can f -stretch the n -bit DFA FA_0 into NFA FA_1 in the following way:

- FA_1 is an $\frac{n}{f}$ -bit NFA.
- There is a bijection between Σ_0 and Σ_1^f ie. for every bit string of length n in Σ_0 there is a sequence of f bit strings of length $\frac{n}{f}$ in Σ_1^f and vice versa.
- For every transition in FA_0 there are f sequential transitions in FA_1 , obeying the above bijection between Σ_0 and Σ_1^f for the labels of the transitions.

Of course, this specialization of stretching is only allowed if n is divisible by f . In that case we call the DFA ‘ f -stretchable’. Again, jamming is the opposite transformation: if an n -bit NFA is ‘ f -jammable’, the resulting automaton is an nf -bit DFA.

EXAMPLE 4.2. To illustrate the stretching of n -bit automata we give an example. The 2-bit DFA FA_0 of figure 5 can be stretched by a factor 2 into the 1-bit NFA FA_1 of figure 7. Also, the 1-bit NFA FA_1 can be jammed into the 2-bit DFA FA_0 . Furthermore, NFA FA_1 can be determinized into DFA FA_2 of figure 9.

4.2 Main results

In this section we will prove a number of propositions about stretching and jamming. From these propositions we can draw conclusions about when stretching or jamming would be useful in terms of memory consumption and string recognition time.

The first four propositions will deal with bit-level stretching. For those first four propositions we assume that n -bit DFA $M = (Q, \Sigma, \delta, q_0, F)$ with the number of states $r = |Q|$, the alphabet size $s = |\Sigma|$ and the number of transitions t , is stretched by a factor f into $\frac{n}{f}$ -bit NFA $M' = (Q', \Sigma', \delta', q'_0, F')$, with $Q' = S \cup I$, $r' = |Q'|$, $s' = |\Sigma'|$ and t' the number of transitions. The number of additional intermediate states in NFA M' is $I = r' - r$. The transition table sizes are respectively rs and $r's'$.

The following propositions hold:

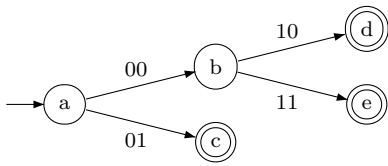


Figure 5. DFA FA_0

	00	01	10	11
a	{b}	{c}	-	-
b	-	-	{d}	{e}
c	-	-	-	-
d	-	-	-	-
e	-	-	-	-

Figure 6. Transition table of DFA FA_0

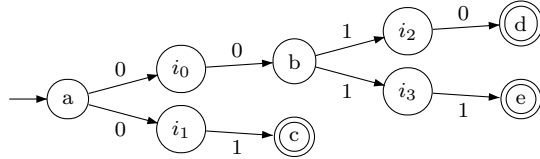


Figure 7. NFA FA_1 , a '2-stretch' of DFA FA_0

	0	1
a	{ i_0, i_1 }	-
i_0	{b}	-
i_1	-	{c}
b	-	{ i_2, i_3 }
i_2	{d}	-
i_3	-	{e}
c	-	-
d	-	-
e	-	-

Figure 8. Transition table of NFA FA_1

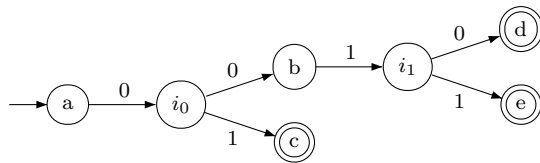


Figure 9. DFA FA_2 , determinized NFA FA_1

	0	1
a	{ i_0 }	-
i_0	{b}	{c}
b	-	{ i_1 }
i_1	{d}	{e}
c	-	-
d	-	-
e	-	-

Figure 10. Transition table of DFA FA_2

PROPOSITION 4.3. $s' = \sqrt[f]{s}$

PROOF. If M is an n -bit DFA, then the alphabet size, s , is 2^n . If M is stretched by a factor f into M' , M' is an $\frac{n}{f}$ -bit NFA, so the alphabet size, s' , is $2^{\frac{n}{f}} = \sqrt[f]{2^n} = \sqrt[f]{s}$. \square

PROPOSITION 4.4. $r' = t(f - 1) + r$

PROOF. Stretching by a factor f introduces $f - 1$ additional intermediate states, for each single transition in the original DFA. Therefore r' is equal to the additional intermediate states, $t(f - 1)$, plus the number of states in the original DFA, r . \square

PROPOSITION 4.5. $r \leq r' \leq rs(f - 1) + r$

PROOF. From proposition 4.4 we know that if DFA M , with t transitions, is stretched by a factor f into M' , $r' = t(f - 1) + r$. The number of transitions is at least 0 and at most rs . Therefore, $r \leq r' \leq rs(f - 1) + r$. \square

PROPOSITION 4.6. Let $z = \frac{r(s - \sqrt[f]{s})}{\sqrt[f]{s}(f - 1)}$.

- $t < z \Leftrightarrow r's' < rs$
- $t = z \Leftrightarrow r's' = rs$
- $t > z \Leftrightarrow r's' > rs$

PROOF. From proposition 4.3 we know that if DFA M is stretched by a factor f to into M' , $s' = \sqrt[f]{s}$. From proposition 4.4 we know that if DFA M , with t transitions, is stretched by a factor f to into M' , $r' = t(f - 1) + r$. Thus, $r's' = (t(f - 1) + r) \sqrt[f]{s}$.

If $t = z = \frac{r(s - \sqrt[f]{s})}{\sqrt[f]{s}(f - 1)}$ then:

$$\begin{aligned}
 r's' &= \left(\frac{r(s - \sqrt[f]{s})}{\sqrt[f]{s}(f - 1)}(f - 1) + r \right) \sqrt[f]{s} \\
 &= rs
 \end{aligned}$$

The inequalities follow directly by similar reasoning. \square

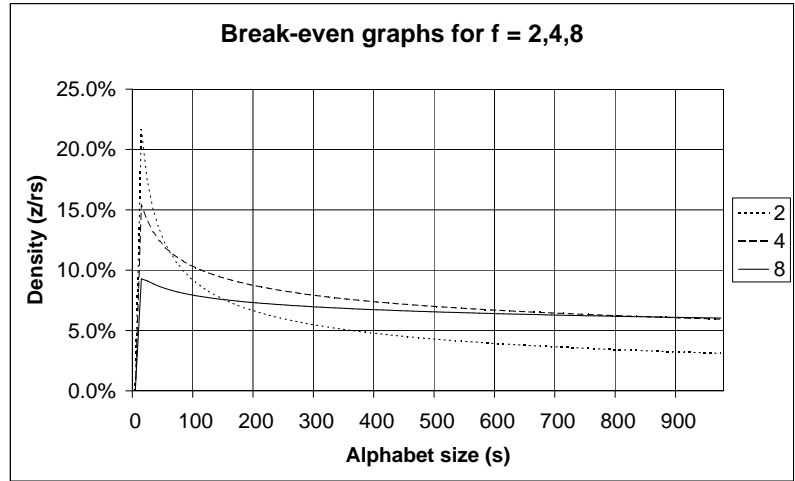


Figure 11. Break-even graphs for stretching

From proposition 4.6 we can conclude that bit-stretching a DFA reduces the transition table size when $t < z$. Therefore it can reduce the amount of memory needed for a transition table representation.

The *transition density* of an automaton is the number of transitions divided by the transition table size (the maximum number of possible transitions), $\frac{t}{rs}$. The density needed to get an equal transition table size after stretching a DFA (the break-even point) is $\frac{z}{rs} = \frac{1-s^{-\frac{1}{f}}}{1-f}$. This value is not dependent on the number of states, r . In figure 11, $\frac{z}{rs}$ is set out against the alphabet size s for different values of f . From these graphs we conclude that the transition density has to be very low to obtain a smaller transition table size by stretching.

The next three propositions will deal with bit-level jamming. For these propositions we assume that n -bit NFA $M = (Q, \Sigma, \delta, q_0, F)$ with $Q = S \cup R$, the number of states $r = |Q|$, the alphabet size $s = |\Sigma|$ and the number of transitions t , is jammed by a factor f into nf -bit DFA $M' = (Q', \Sigma', \delta', q'_0, F')$, with $r' = |Q'|$, $s' = |\Sigma'|$ and t' the number of transitions. The number of redundant intermediate states in NFA M is $R = r - r'$. The transition table sizes are respectively rs and $r's'$.

PROPOSITION 4.7. $s' = s^f$

PROOF. If M is an n -bit NFA, then the alphabet size, s , is 2^n . If M is jammed by a factor f into M' , M' is an nf -bit DFA, so the alphabet size, s' , is $2^{nf} = (2^n)^f = s^f$. □

PROPOSITION 4.8. $0 \leq r' \leq r$

PROOF. Every automaton has at least 0 states, thus $r' \geq 0$. Jamming removes all redundant intermediate states so $r' \leq r$. □

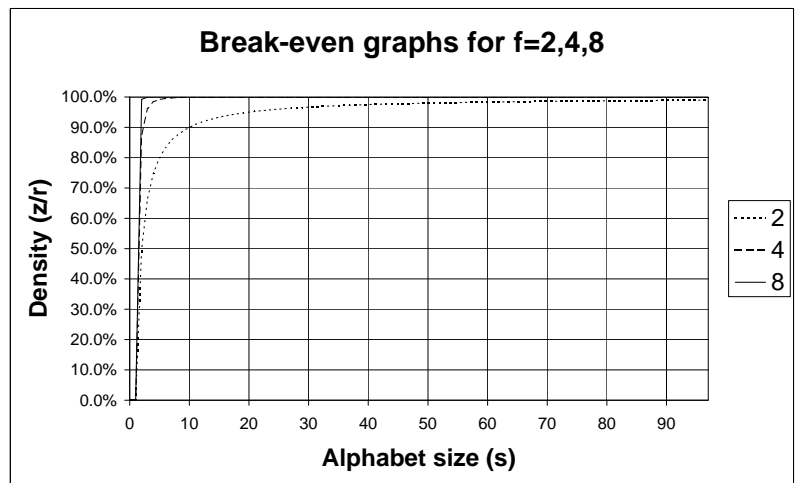


Figure 12. Break-even graphs for jamming

PROPOSITION 4.9. Let $z = r - \frac{r}{s^{f-1}}$, and $r - r'$ the number of redundant intermediate states.

$$\begin{aligned} r - r' > z &\Leftrightarrow r's' < rs \\ r - r' = z &\Leftrightarrow r's' = rs \\ r - r' < z &\Leftrightarrow r's' > rs \end{aligned}$$

PROOF. From proposition 4.7 we know that if NFA M is jammed by a factor f to into M' then $s' = s^f$. If M is jammed into M' then $r - r'$ states are removed. So if $r - r' = r - \frac{r}{s^{f-1}}$ then $r' = r - (r - \frac{r}{s^{f-1}})$. Therefore, $r's' = \frac{r}{s^{f-1}}s^f = rs$. Again, the inequalities follow directly by similar reasoning. \square

From these propositions we can conclude that bit-jamming an NFA can potentially reduce the number of states and the number of transitions. Therefore it can reduce the average time needed to recognize a string. The *redundant state density* of an automaton is the number of redundant intermediate states divided by the total number of states, $\frac{r-r'}{r}$. The density needed to get an equal transition table size after jamming an NFA is $\frac{z}{r} = 1 - s^{1-f}$. This value is not dependent on the number of states, r . In figure 12, $\frac{z}{r}$ is set out against the alphabet size s for different values of f . From these graphs we can conclude that the redundant state density has to be very high to obtain a smaller transition table size by jamming. While these theoretical results indicate that jamming will often require a larger transition table, we expect that it will reduce the average string recognition time. However, this has to be examined in practice.

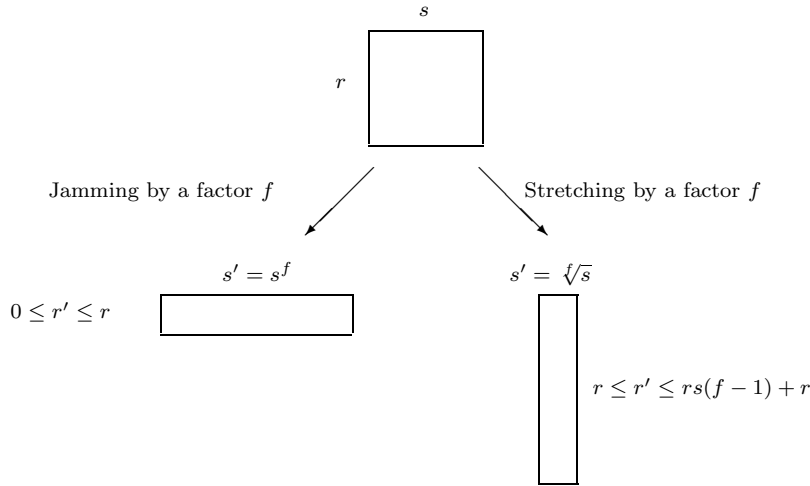


Figure 13. Stretching and Jamming

EXAMPLE 4.10. Figure 13 visually represents how the dimensions of the transition tables change, by virtue of propositions 4.3, 4.5, 4.7 and 4.8.

5. ALGORITHMS

In this section we present three algorithms, one for stretching, one to determine if an NFA is jammable and one for jamming. First, we need some definitions. We use **mod** as the operator for the remainder of integer division.

In our stretching algorithm we conceive of alphabet elements as strings of subelements (typically bit substrings). If alphabet element $a \in \Sigma$ has length $|a|$ then we number the subelements $a.0, \dots, a.(|a| - 1)$. Thus, if $a = 0111$ then $a.0 = 0, a.1 = 1, a.2 = 1$ and $a.3 = 1$. We use square brackets to denote a substring of an alphabet element. The length of the substring depends on the factor f used in stretching. For $a \in \Sigma, a[i] = a.(i \lfloor \frac{|a|}{f} \rfloor \dots (i + 1) \lfloor \frac{|a|}{f} \rfloor - 1)$. So for example, if FA_0 is stretched by a factor 2 and $a = 0111$, then $a[0] = 01$ and $a[1] = 11$. If FA_0 is stretched by a factor 4 then $a[0] = 0, a[1] = 1, a[2] = 1$ and $a[3] = 1$.

PROPOSITION 5.1. Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be an n -bit DFA. Algorithm 5.2 will stretch FA_0 by a factor f into $\frac{n}{f}$ - bit NFA $FA_1 = (S_1, \Sigma_1, \delta_1, q_1, F_1)$.

ALGORITHM 5.2. **Stretch**(FA_0, FA_1, f):

Pre : $n \in \mathbb{Z}^+ \wedge f \in \mathbb{Z}^+ \setminus \{1\} \wedge n \bmod f = 0$
Post : FA_1 is an f -stretch of FA_0

```

[[ S1 := S0
; q1 := q0
; F1 := F0
; Σ1 := 'all bit strings of length  $\frac{n}{f}$ '
; for all ((p, a), q) : ((p, a), q) ∈ δ0 →
    I := I ∪ {spq0, ..., spqf-2}
    ; δ1 := δ1 ∪ {(p, a[0]), spq0}
    ; for i := 0 to f - 3 → δ1 := δ1 ∪ {(spqi, a[i + 1]), spqi+1} rof
    ; δ1 := δ1 ∪ {(spqf-2, a[f - 1]), q}
rof
]]

```

PROOF. We need to prove that the algorithm will stretch FA_0 according to the conditions in definition 3.3.

- Each bit string of length n from Σ_0 can be constructed by a sequence of f bit strings of length $\frac{n}{f}$ from Σ_1 . Note that this sequence is unique for every bit string from Σ_0 . Furthermore, because Σ_0 consists of all bit strings of length n , every sequence of f bit strings of length $\frac{n}{f}$ from Σ_1 forms a unique bit string from Σ_0 . Therefore, there is a bijection between Σ_0 and Σ_1^f .
- After the algorithm is executed, S_0 is equal to S_1 , q_0 is equal to q_1 and F_0 is equal to F_1 , therefore there is an obvious bijection between S_0 and S_1 . For each transition (p, a, q) in FA_0 the algorithm creates an unbranched path of length f from p to q in FA_1 , with $a[0], \dots, a[f - 1]$ as labels so the conditions of definition 3.3 hold.

□

In jamming, we only consider NFAs in which all states are on a path from the start state and all states are on a path to at least one of the final states. In section 3 we showed that not every NFA is jammable so first we will give the properties that are needed for an NFA to be jammable. After that we will present an algorithm to determine if an NFA is jammable. For reasons of simplicity we only give the properties for an NFA to be jammable by a factor 2, and our algorithm will also only determine if an NFA is jammable by a factor 2. This algorithm can be generalized to an algorithm that determines if an NFA is jammable by a factor f .

Both algorithms below are based on the breadth-first search algorithm and make use of a fifo queue Q and operations *queue* and *dequeue*. These are standard notions (see for example [Cormen et al. 2001]).

The two algorithms could also be combined into one algorithm so that an NFA is either jammed or the algorithm returns an indication that the NFA is not jammable by a factor of 2.

PROPOSITION 5.3. *Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be an n -bit NFA. FA_0 is jammable by a factor 2 iff it has the following two properties:*

- (1) *The states in the transition graph can be two-coloured, so the states can be partitioned into two sets:
An 'even' set E . The lengths of all paths from q_0 to a state in E are even.
An 'odd' set O . The lengths of all paths from q_0 to a state in O are odd.*
- (2) *All final states are in set E .*

PROOF. First we prove that if FA_0 has properties 1 and 2, it is jammable by a factor 2:

Choose an arbitrary state $q \in O$. Because of property 1, $q \neq q_0$ and q cannot have a transition to itself. From property 2 we can conclude that $q \notin F_0$. Because there must be a path from q_0 to q and from q to a state $f \in F_0$, q must at least have one in-going transition and one out-going transition. Let the in-going transitions be $(p_0, a_0, q), (p_1, a_1, q), \dots, (p_i, a_i, q)$, and let the out-going transitions be $(q, b_0, r_0), (q, b_1, r_1), \dots, (q, b_j, r_j)$. Now FA_0 can be jammed in the following way: remove state q and all its in-going and out-going transitions and create new transitions $(p_0, a_0b_0, r_0), (p_0, a_0b_1, r_1), \dots, (p_0, a_0b_j, r_j), (p_1, a_1b_0, r_0), \dots, (p_i, a_ib_j, r_j)$. This process can be repeated until all states from set O are removed.

Now we prove that FA_0 is not jammable if it does not have either property 1 or 2:

If FA_0 lacks property 1 there must be a state p that cannot be partitioned into either set. Therefore there must be a path of even length and a path of odd length from q_0 to p . From this we can conclude that there must be a path of odd length from q_0 to a final state. It is straightforward to see that it is impossible to jam a path of odd length by a factor of 2.

If FA_0 lacks property 2 then there is a final state f in set O . Therefore there is a path of odd length from q_0 to f . Again, it is impossible to jam a path of odd length by a factor of 2. □

PROPOSITION 5.4. Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be an n -bit NFA. Algorithm 5.5 will determine if FA_0 is jammable by a factor 2.

ALGORITHM 5.5. **Jammable**(FA_0, Q, E, O):

```

Pre :  $n \in \mathbb{Z}^+ \wedge n \bmod 2 = 0$ 
Post : The algorithm will return if  $FA_0$  is jammable by a factor 2 or not
[[  $Q := E := O := \emptyset$ 
;  $E := E \cup \{q_0\}$ 
;  $enqueue(q_0, Q)$ 
do  $Q \neq \emptyset \rightarrow$ 
   $p := dequeue(Q)$ 
; for all  $q, a : ((p, a), q) \in \delta_0 \rightarrow$ 
  if  $q \notin E \cup O \wedge p \in E \wedge q \notin F_0 \rightarrow O := O \cup \{q\}; enqueue(q, Q)$ 
  ||  $q \notin E \cup O \wedge p \in E \wedge q \in F_0 \rightarrow$  return ' $FA_0$  is not jammable by a factor 2'
  ||  $q \notin E \cup O \wedge p \in O \rightarrow E := E \cup \{q\}; enqueue(q, Q)$ 
  ||  $q \in E \cup O \wedge (p \in E \wedge q \in E) \vee (p \in O \wedge q \in O) \rightarrow$  return ' $FA_0$  is not jammable by a factor 2'
  fi
  rof
od
; return ' $FA_0$  is jammable by a factor 2'
]]

```

PROOF. The algorithm is based on the BFS algorithm, see for example [Cormen et al. 2001]. The BFS algorithm ensures that if a new state q is found, the path traveled by the algorithm from q_0 to q is the shortest path. The algorithm searches for all states q that have an incoming transition from a found state p .

If $q \notin E \cup O$ then q has not been found yet. This means that it can be added to the appropriate set E or O unless $p \in E$ and $q \in F_0$. If this is the case then q must be added to O which would violate property 2 of proposition 5.3.

If $q \in E \cup O$ then q has already been found. If $p \in E$ then q must be in O , otherwise there is a violation of property 1 of proposition 5.3. Analogously, if $p \in O$ then q must be in E . \square

PROPOSITION 5.6. Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be an n -bit NFA. Algorithm 5.7 will jam FA_0 by a factor 2 into $2n$ -bit DFA $FA_1 = (S_1, \Sigma_1, \delta_1, q_1, F_1)$ iff FA_0 is jammable.

ALGORITHM 5.7. **Jam**(FA_0, FA_1, Q, V):

```

Pre :  $n \in \mathbb{Z}^+ \wedge n \bmod 2 = 0$ 
Post :  $FA_1$  is a 2-jam of  $FA_0$ 
[[  $Q := V := \emptyset$ 
;  $S_1 := S_0$ 
;  $q_1 := q_0$ 
;  $F_1 := F_0$ 
;  $\Sigma_1 :=$  'all bit strings of length  $2n$ '
;  $enqueue(q_0, Q)$ 
do  $Q \neq \emptyset \rightarrow$ 
   $p := dequeue(Q)$ 
; for all  $q, a : ((p, a), q) \in \delta_0 \rightarrow$ 
  for all  $r, b : ((q, b), r) \in \delta_0 \rightarrow$ 
     $S_1 := S_1 \setminus \{q\}$ 
    ;  $\delta_1 := \delta_1 \cup \{((p, ab), r)\}$ 
  ; if  $r \notin V \rightarrow$ 
     $V := V \cup \{r\}$ 
    ;  $enqueue(r, Q)$ 
  fi
  rof
rof
rof

```

od

]]

PROOF. To show that FA_1 is a 2-jam of FA_0 we have to prove that FA_0 is a 2-stretch of FA_1 according to definition 3.4.

- Each bit string of length $2n$ from Σ_1 can be constructed by a sequence of 2 bit strings of length n from Σ_0 . Note that this sequence is unique for every bit string from Σ_1 . Furthermore, because Σ_1 consists of all bit strings of length $2n$, every sequence of 2 bit strings of length n from Σ_0 forms a unique bit string from Σ_1 . Therefore, there is a bijection between Σ_1 and Σ_0^2 .
- After the algorithm is executed, S_1 is equal to S_0 , q_1 is equal to q_0 and F_1 is equal to F_0 , therefore there is an obvious bijection between S_1 and S_0 . Because FA_0 is jammable, the states can be divided in the ‘even’ set E and the ‘odd’ set O according to proposition 5.3. The algorithm is based on the BFS algorithm and searches for all states p from set E . From state p it searches for all paths $\langle (p, a, q), (q, b, r) \rangle$ of length 2 in FA_0 , removes the redundant state q from S_1 and constructs the transition (p, ab, r) in δ_1 . Therefore, for each transition (p, ab, r) in FA_1 there is a path of length 2 from p to r in FA_0 with a and b as labels so the conditions of definition 3.4 hold.

□

6. CONCLUSIONS AND FUTURE WORK

We have defined the notions of stretching and jamming and shown the theoretical conditions under which they reduce the amount of memory needed for a transition table. In the case of stretching, these conditions seem to be the most practical. We also have also suggested that jamming can potentially reduce the string recognition time of an NFA, in that fewer transitions need to be made.

Empirical research of stretching and jamming is now clearly required. Relying on the algorithms that have been given above, empirical measurements of memory and time requirements will be made for various stretching and jamming scenarios. The objective will be to discover the practical conditions under which stretching and jamming can be used to maximum advantage.

Another direction that can be investigated is stretching and jamming of regular expressions.

REFERENCES

- AHO, A., SETHI, R., AND ULLMAN, J. 1986. *Compilers : principles, techniques and tools*. Addison-Wesley.
- CORMEN, T., LEISERSON, C., AND RIVEST, R. 2001. *Introduction to algorithms*. McGraw-Hill.
- HOPCROFT, J., MOTWANI, R., AND ULLMAN, J. 2001. *Introduction to automata theory, languages, and computation*. Addison-Wesley.
- WATSON, B. 1995. Taxonomies and toolkits of regular language algorithms. Ph.D. thesis, Eindhoven University of Technology, The Netherlands.