Stretching and Jamming of Finite Automata

Noud de Beijer∗ Derrick G. Kourie† Bruce W. Watson∗†

Abstract

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 processing time. Given a 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 inverse 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 and we will calculate theoretical bounds, when stretching/jamming is effective in terms of memory consumption.

1 Introduction

In this paper we present two new transformations on automata. A classical application area of automata theory is compiler construction. 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 [ASU86, 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 [ASU86, p. 144]. The lexical analyzer uses finite automata to recognize languages. This finite automaton uses a transition function to process strings but there are different ways to implement this transition function.

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 [ASU86, p. 114]. Of course, next to compilers there are numerous other applications in computing science where automata are used.

Thus, transformations on automata that increase their performance in terms of memory consumption or string processing time are potentially useful (see for example [Wat95]). We propose two transformations on automata: stretching and jamming. Under certain conditions, these transformations will produce more efficient automata in terms of memory consumption and string processing time.

∗Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
†Department of Computer Science, University of Pretoria, Pretoria 0002, South Africa
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-bit characters.

Jamming is the inverse transformation, in which two successive transitions (based on, for example, input characters represented in 8-bits) are transformed into a single transition. This single transition will then be based on an input character represented by 16-bits. The same transformations can be used on a nondeterministic finite automaton (NFA).

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 [HMU01]) 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, $\Sigma$ is the alphabet, $\delta : Q \times \Sigma \to Q$ 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.

The $n$-closure of an alphabet is the set of all symbols that consist of concatenating $n$ symbols from $\Sigma$. $\Sigma^+$ is the plus-closure of the alphabet, the set of symbols obtained by concatenating one or more symbols from $\Sigma$.

$|Q|\Sigma$ is the theoretical 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 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. Preliminary benchmarking results indicate that our theoretical transition table size is indeed a good estimate for the real transition table size.

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 $q = \delta(p, a)$. We will also use the notation $(p, a, q) \in \delta$.

A path of length $k$ in a DFA $M$ is a sequence $\langle (r_0, a_0, r_1), \ldots, (r_{k-1}, a_{k-1}, r_k) \rangle$, 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 string, or word $a_0a_1\cdots a_{k-1} \in \Sigma^k$ is the label of the path.

The extended transition function of a DFA $M$, $\hat{\delta} : Q \times \Sigma^+ \to Q$, 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: $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function. Note that $\mathcal{P}(Q)$ is the powerset of $Q$. For present purposes, $\varepsilon$-transitions can be ignored without loss of generality.

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 $\langle (r_0, a_0, r_1), \ldots, (r_{k-1}, a_{k-1}, r_k) \rangle$, 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 $0 \leq i < k$. 2
In an NFA $M$, the extended transition function, $\delta : Q \times \Sigma^+ \to \mathcal{P}(Q)$, is also defined so that $r_j \in \delta(r_i, w)$ iff there is a path from $r_i$ to $r_j$, labeled $w$.

To stretch a transition we need to split up a single symbol in 2 or more sub-symbols. Therefore, 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 $a.0, \ldots, a.(|a| - 1)$. Thus, if $a = 0111$ then $a.0 = 0$, $a.1 = 1$, $a.2 = 1$ and $a.3 = 1$.

By definition, a word is a string of symbols over an alphabet. We use the same notation to number the individual symbols of a word. So, for the word $w \in \Sigma^k$, we number the individual symbols $w.0 \cdots w.(k - 1)$.

Because in our paper it will always be clear whether $w$ is a word or a single symbol, no conflicts will rise because of this definition.

### 3 Formal Definitions

#### 3.1 General Definitions

In this section we give formal definitions of the stretching and jamming operations. One way in which we can stretch an automaton is by transforming each transition into $k$ sequential transitions. This stretching operation on a single transition is pictured in figure 1.

![Figure 1: Stretching transition $(p, a, q)$ into $k$ sequential transitions.](image)

In this example we see that transition $(p, a, q)$ is stretched into $k$ sequential transitions and $k - 1$ new states are introduced. In this sequence of transitions we call $p$ and $q$ the original states, and $i_0, \ldots, i_{k-2}$ the additional intermediate states.

Jamming is the inverse transformation, in which $k$ sequential transitions are transformed into a single transition. In figure 1 this can be seen as performing a transformation in the opposite direction to the stretching operation.

This means that the intermediate states are removed. In the case of jamming we call these states redundant intermediate states.

We can stretch NFAs as well as DFAs. We will define the stretching transformation on NFAs, and the result of a transformation will also be an NFA. Because DFAs are a subset of NFAs, the stretching of DFAs is automatically defined. If we stretch a DFA, in some cases the resulting automaton may have more than one transition with the same label from a given state and therefore
the result of stretching a DFA might be an NFA. In section 3.2 we will present an example that will clarify this.

If NFA FA₀ can be stretched into NFA FA₁, we call FA₁ a stretch of FA₀. The set of states of FA₁ consists of a subset S₁ of original states and a subset I of newly introduced additional intermediate states. Definition 3.1 formally describes a general stretching transformation.

Firstly, there is an injection \( \tau \) from the alphabet of FA₀, \( \Sigma_0 \) to \( \Sigma_1^+ \), the plus-closure of the alphabet of FA₁ (property 1). Secondly, there is a one-to-one relation \( \varphi \) between the original states of FA₀ and the original states of FA₁.

This bijection connects the start states of FA₀ and FA₁ (property 2). It also defines a one-to-one relationship between the final states of both automata (property 3). Property 4 states that for every transition from state \( p \) to \( q \) with label \( a \) in FA₀ there exists a path from \( \varphi(p) \) to \( \varphi(q) \) with label \( \tau(a) \) in FA₁, which travels from a state in \( S_1 \) via a number of intermediate states to another state in \( S_1 \). The inverse of this property is also true, therefore property 5 also holds.

**Definition 3.1.** Let \( FA₀ = (S₀, \Sigma₀, δ₀, q₀, F₀) \) be an NFA, and let \( FA₁ = (S₁ \cup I, \Sigma₁, δ₁, q₁, F₁) \) be an NFA. \( FA₁ \) is a stretch of \( FA₀ \) iff:

- There is an injection \( \tau : \Sigma₀ → \Sigma₁^+ \), thus:
  \[
  (\forall a₀, a₁ : a₀, a₁ ∈ \Sigma₀ : \tau(a₀) = \tau(a₁) ⇒ a₀ = a₁) \tag{1}
  \]

- There is a bijection \( \varphi : S₀ ↔ S₁ \), with the following properties:
  \[
  - \varphi(q₀) = q₁
  \]
  \[
  - (\forall f₀ ∈ F₀ : (\exists f₁ ∈ F₁ : \varphi(f₀) = f₁)) \land |F₀| = |F₁| \tag{2}
  \]
  \[
  - (\forall p, a, q : q ∈ \delta₀(p, a) : (\exists k, r₀, . . . , rₖ, w : rₖ ∈ \hat{δ₁}(r₀, w) : \varphi(p) = r₀ \land \varphi(q) = rₖ \land \tau(a) = w)) \tag{3}
  \]
  \[
  - (\forall k, r₀, . . . , rₖ, w : rₖ ∈ \hat{δ₁}(r₀, w) : \varphi(rₖ) ∈ \delta₁(\varphi(r₀), \tau⁻¹(w))) \tag{4}
  \]
  where \( p, q ∈ S₀, a ∈ \Sigma₀, r₀, rₖ ∈ S₁, r₁, . . . , rₖ⁻¹ ∈ I, \) and \( w ∈ \Sigma₁^+ \) for \( k ≥ 1 \). Furthermore, \( \hat{δ₁}(rᵢ, w, i) = rᵢ₊₁ \), for \( 0 ≤ i < k \).

We define jamming as the inverse transformation of stretching. Because of symmetry, if we jam certain NFAs the result will be a DFA.

If NFA FA₀ is jammed into NFA FA₁ (FA₁ is a jam of FA₀) then FA₀ is a stretch of FA₁. The set of states of FA₀ consists of a subset \( S₀ \) of original states and a subset \( R \) of redundant intermediate states. These redundant intermediate states will be removed by the jamming transformation.

**Definition 3.2.** Let \( FA₀ = (S₀ ∪ R, \Sigma₀, δ₀, q₀, F₀) \) be an NFA, and let \( FA₁ = (S₁, \Sigma₁, δ₁, q₁, F₁) \) be an NFA. \( FA₁ \) is a jam of \( FA₀ \) iff \( FA₀ \) is a stretch of \( FA₁ \), with \( R \) being the set of additional intermediate states, resulting from stretching.

### 3.2 Stretching and Jamming by a Factor \( f \)

In the previous section we presented definition 3.1. This definition is very general: every single transition can be stretched into a different number of sequential transitions, according to properties 4 and 5.
In the next section our major application of stretching and jamming will be introduced. To be able to use the definitions in that application, we need to make a restriction on the previous definition. In the definition below we only allow all transitions to be stretched into a fixed number of sequential transitions.

Therefore, we introduce the factor $f$ in stretching and jamming. If NFA $FA_0$ is stretched by a factor $f$ into NFA $FA_1$, we call $FA_1$ an $f$-stretch of $FA_0$. This means that the relation $\tau$ specializes to a one-to-one relationship between the alphabet of $FA_0$ and the $f$-closure of the alphabet of $FA_1$. Furthermore, for each transition in $FA_0$ there are exactly $f$ sequential transitions in $FA_1$.

The formal differences between the new definition and the previous definition are expressed in properties 7, 10 and 11. In this definition, the relation $\tau$ is not only an injection, but because of property 7 also a surjection, and therefore a bijection. Also, because of property 10, for every transition $(p,a,q)$ in the original NFA, there is a path of length $f$ from $\varphi(p)$ to $\varphi(q)$ with label $\tau(a)$. Property 11 states that the inverse is also true.

**Definition 3.3.** Let $FA_0 = (S_0, \Sigma_0, \delta_0, q_0, F_0)$ be an NFA, 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 $\tau : \Sigma_0 \leftrightarrow \Sigma_1^f$, thus:
  
  \begin{align*}
  & \forall a_0, a_1 : a_0, a_1 \in \Sigma_0 : \tau(a_0) = \tau(a_1) \Rightarrow a_0 = a_1 \quad (6) \\
  & \forall w \in \Sigma_1^f : (\exists a \in \Sigma_0 : \tau(a) = w) \quad (7)
  \end{align*}

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

  \begin{align*}
  & \varphi(q_0) = q_1 \quad (8) \\
  & \forall f_0 \in F_0 : (\exists f_1 \in F_1 : \varphi(f_0) = f_1) \land |F_0| = |F_1| \quad (9) \\
  & \forall p, a, q : q \in \delta_0(p, a) : (\exists r_0, \ldots, r_f, w : r_f \in \delta_1(r_0, w) : \varphi(r_0) = p \land \varphi(r_f) = q \land \tau(a) = w) \quad (10) \\
  & \forall r_0, \ldots, r_f, w : r_f \in \delta_1(r_0, w) : \varphi(r_f) \in \delta_0(\varphi(r_0), \tau^{-1}(w)) \quad (11)
  \end{align*}

where $p, q \in S_0$, $a \in \Sigma_0$, $r_0, r_f \in S_1$, $r_1, \ldots, r_{f-1} \in I$, and $w \in \Sigma_1^f$ for $f \geq 2$.

Furthermore, $\delta_1(r_i, w, i) = r_{i+1}$, for $0 \leq i < f$.

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 an NFA $FA_0$ which can be stretched into $FA_0$.

**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 an NFA. $FA_1$ is an $f$-jam of $FA_0$ iff $FA_0$ is an $f$-stretch of $FA_1$. The set $R$ of $FA_0$ is the set of additional intermediate states, resulting from stretching.

To illustrate these definitions we give an example:

**Example 3.5.** The graph in figure 2 represents the DFA $FA_0 = (\{q, r, s\}, \{a, b, c, d\}, \delta_1, q, \{s\})$. DFA $FA_0$ can be stretched by a factor 2 into NFA $FA_1$ of figure 3. NFA $FA_1 = (S_2 \cup I, \{e, f, g, h, i, j, k\}, \delta_2, t, \{y\})$, with
Injection $\tau$ and bijection $\varphi$ are shown in figures 4 and 5 respectively.

In this example we can see why stretching a DFA can result in an NFA. Because we chose $\tau(b) = gh$ and $\tau(c) = gi$, there are two outgoing transitions with label $g$ from state $w$. Therefore $FA_1$ is an NFA.

4 Bit-level Stretching and Jamming

4.1 Overview

In order to highlight how stretching and jamming can improve performance, we present an application of both transformations in this section. 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 $\Sigma_0$ and $\Sigma_1^f$, i.e. for every bit string of length $n$ in $\Sigma_0$ there is a sequence of $f$ bit strings of length $\frac{n}{f}$ in $\Sigma_1^f$ and vice versa.
- For every transition in $FA_0$ there are $f$ sequential transitions in $FA_1$, obeying the above bijection between $\Sigma_0$ and $\Sigma_1^f$ 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 inverse 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 6 can be stretched by a factor 2 into the 1-bit NFA \( FA_1 \) of figure 8. 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 10.

Note that in the previous example, we stretched a transition by using the most significant bit first. For example, we stretched transition \((a, 01, c)\) into \((a, 0, i_1)\) and \((i_1, 1, c)\), taking the 0 first and then the 1. In practice, we will usually use the least significant bit first, because that is the most natural way to process a bit string. It can be done as presented here in practice too, however.
4.2 Theoretical 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 will be useful in terms of memory consumption and string processing time. The first four propositions will deal with bit-level stretching. The following propositions hold:

**Proposition 4.3.** \( | \Sigma_1 | = \sqrt{| \Sigma_0 |} \)

*Proof.* If \( \text{FA}_0 \) is an \( n \)-bit NFA, then the alphabet size, \(| \Sigma_1 |\), is \( 2^n \). If \( \text{FA}_0 \) is stretched by a factor \( f \) into \( \text{FA}_1 \), \( \text{FA}_1 \) is an \( \frac{n}{f} \)-bit NFA, so the alphabet size, \(| \Sigma_1 |\), is \( 2^{\frac{n}{f}} = \sqrt{2^n} = \sqrt{| \Sigma_0 |} \).

**Proposition 4.4.** \( |Q_1| = |\delta_0|(f - 1) + |Q_0| \)

*Proof.* Stretching by a factor \( f \) introduces \( f - 1 \) additional intermediate states, for each single transition in the original DFA. Therefore \(|Q_1|\) is equal to the additional intermediate states, \(|\delta_0|(f - 1)\), plus the number of states in the original DFA, \(|Q_0|\).

**Proposition 4.5.** \( |Q_0| \leq |Q_1| \leq |Q_0| |\Sigma_1| (f - 1) + |Q_0| \)

*Proof.* From proposition 4.4 we know that if DFA \( \text{FA}_0 \), with \(|\delta_0|\) transitions, is stretched by a factor \( f \) into \( \text{FA}_1 \), \(|Q_1| = |\delta_0|(f - 1) + |Q_0|\). The number of transitions is at least 0 and at most \(|Q_0| |\Sigma_0|\). Therefore, \(|Q_0| \leq |Q_1| \leq |Q_0| |\Sigma_1| (f - 1) + |Q_0| \).

**Proposition 4.6.** Let \( z = \frac{|Q_0||\Sigma_0| - \sqrt{|\Sigma_0|}}{\sqrt{|\Sigma_0|}(f - 1)} \).

\[ |\delta_0| < z \iff |Q_1||\Sigma_1| < |Q_0||\Sigma_0| \]
\[ |\delta_0| = z \iff |Q_1||\Sigma_1| = |Q_0||\Sigma_0| \]
\[ |\delta_0| > z \iff |Q_1||\Sigma_1| > |Q_0||\Sigma_0| \]

*Proof.* From proposition 4.3 we know that if DFA \( \text{FA}_0 \) is stretched by a factor \( f \) into \( \text{FA}_1 \), \(|\Sigma_1| = \sqrt{|\Sigma_0|}\). From proposition 4.4 we know that if DFA \( \text{FA}_0 \), with \(|\delta_0|\) transitions, is stretched by a factor \( f \) to into \( \text{FA}_1 \), \(|Q_1| = |\delta_0|(f - 1) + |Q_0|\). Thus, \(|Q_1||\Sigma_1| = (|\delta_0|(f - 1) + |Q_0|) \sqrt{|\Sigma_0|}\). Therefore, if \( |\delta_0| = z = |Q_0||\Sigma_0| - \sqrt{|\Sigma_0|} \) then:

\[ |Q_1||\Sigma_1| = (|Q_0||\Sigma_0| - \sqrt{|\Sigma_0|}) / \sqrt{|\Sigma_0|} = |Q_0||\Sigma_0| \]

The inequalities follow directly by similar reasoning.

From proposition 4.6 we can conclude that bit-stretching a DFA reduces the transition table size when \( |\delta_0| < z \). 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). For our DFA \( \text{FA}_0 \) this is \( |\delta_0|/|Q_0||\Sigma_0|\). In the last proposition we saw that if the number of transitions \(|\delta_0|\) is equal to \( z \), the transition table size is the same before and after stretching. This means that the density in that case is \( z/|Q_0||\Sigma_0| = \)
\[
\left(1 - \frac{|\Sigma_0| - f}{|\Sigma_0|}\right)/(1 - f).
\]
This value is not dependent on the number of states, \(|Q_0|\). In figure 12, \(z/|Q_0||\Sigma_0|\) is set out against the alphabet size \(|\Sigma_0|\) for different values of \(f\). From these graphs we can conclude how low the transition density has to be to obtain a smaller transition table size by stretching. For example, if we stretch an 8-bit DFA (256 alphabet symbols) by a factor 2 (\(f=2\)), we reduce the transition table size if the transition density is lower than 6%.

![Break-even graphs for stretching with factors 2, 4, 8.](image)

Figure 12: Break-even graphs for stretching

The next three propositions will deal with bit-level jamming.

**Proposition 4.7.** \(|\Sigma_1| = |\Sigma_0|^f\)

Proof. If \(F_{A_0}\) is an \(n\)-bit DFA, then the alphabet size, \(|\Sigma_0|\), is \(2^n\). If \(F_{A_0}\) is jammed by a factor \(f\) into \(F_{A_1}\), \(F_{A_1}\) is an \(nf\)-bit DFA, so the alphabet size, \(|\Sigma_1|\), is \(2^{nf} = (2^n)^f = |\Sigma_0|^f\). \(\square\)

**Proposition 4.8.** \(0 \leq |Q_1| \leq |Q_0|\)

Proof. Every automaton has at least 0 states, thus \(|Q_1| \geq 0\). Jamming removes all redundant intermediate states so \(|Q_1| \leq |Q_0|\). \(\square\)

We could include a proposition here that states in which cases jamming results in a smaller transition table. Unfortunately, in almost all cases jamming results in a larger transition table. However, one of the reasons for investigating the jamming operation is that it might reduce the string processing time because multiple symbols are processed at once.

This concludes our overview of bit-level stretching and jamming. We end this section with a few notes.

Of course, at this point the question rises how stretching and jamming perform in practice. We have implemented the stretching and jamming operations as algorithms and performed benchmarking studies under different conditions. These benchmarking studies confirm our theoretical results. Furthermore, these
studies show that jamming indeed reduces the string processing time in most cases. The algorithms as well as the results of the benchmarking studies can be found in [dB04].

In this paper, we will only consider stretching or jamming the complete transition table. As we saw in this section, stretching is useful if the transition density is low. Therefore, it might be interesting to stretch only certain parts of the transition table where the transition density is low. For jamming we can argue that if a certain part of the transition table contains many redundant states, jamming this part only might be interesting. We call this approach local stretching and jamming but leave its details as future work.

Furthermore, we will only consider transition tables that can be implemented with a regular matrix that has a row for each state and a column for each input symbol. Of course, there are cases where this does not apply. For example, sometimes character classes or sparse matrices are used for implementing the transition table. We will not discuss these situations in this paper.

As a general guideline however, if a certain transition table implementation leads to a smaller transition table density it will have a positive effect on stretching. If it leads to a higher transition density it will have a negative effect.

5 Conclusions and Future Work

We have defined the notions of stretching and jamming and shown the theoretical conditions under which they influence performance. In the case of stretching, performance can be improved by reducing the memory usage. Jamming on the other hand, increases memory usage.

During the theoretical discussion of jamming we hinted that it might reduce the string processing time but we did not present a theoretical model for that. However, preliminary benchmarking studies in [dB04] confirm our theoretical results and show that jamming indeed reduces the string processing time.

There are a number of interesting problems that can still be investigated further. We only considered stretching or jamming the complete transition table. Transforming only a small part of the transition table, in other words local stretching and jamming, is an interesting problem for further research.

Of course, any method that is used to change the transition table, for example character classes and sparse matrices, influences stretching and jamming. Therefore, these situations can be investigated further.

Lastly, we only looked into transforming automata and not regular expressions. The stretching and jamming of regular expressions is also a candidate for further research.

References

