An Aho-Corasick Based Assessment of Algorithms Generating Failure Deterministic Finite Automata

Madoda Nxumalo[1], Derrick G. Kourie[2,3], Loek Cleophas[2,4], and Bruce W. Watson[2,3]

[1] Computer Science, Pretoria University, South Africa
[2] FASTAR Research, Information Science, Stellenbosch University, South Africa
[3] Centre for Artificial Intelligence Research, CSIR Meraka Institute, South Africa
[4] Foundations of Language Processing, Computer Science, Umeå University, Sweden
{madoda,derrick,loek,bruce}@fastar.org — http://www.fastar.org

Abstract. The Aho-Corasick algorithm derives a failure deterministic finite automaton for finding matches of a finite set of keywords in a text. It has the minimum number of transitions needed for this task. The DFA-Homomorphic Algorithm (DHA) algorithm is more general, deriving from an arbitrary complete deterministic finite automaton a language-equivalent failure deterministic finite automaton. DHA takes formal concepts of a lattice as input. This lattice is built from a state/out-transition formal context that is derived from the complete deterministic finite automaton. In this paper, three general variants of the abstract DHA are benchmarked against the specialised Aho-Corasick algorithm. It is shown that when heuristics for these variants are suitably chosen, the minimality attained by the Aho-Corasick algorithm can be closely approximated. A published non-lattice-based algorithm is also shown to perform well in experiments.

Keywords: Failure deterministic finite automaton, Aho-Corasick algorithm

1 Introduction

A deterministic finite automaton (DFA) defines a set of strings, called its language. It is represented as a graph with symbol-labelled transitions between states. There are efficient algorithms to determine whether an input string belongs to the DFA's language. An approach to reducing DFA memory requirements is the use of so-called failure deterministic finite automata (FDFAs, also defined in Section 2). An FDFA can be used to define the same language as a DFA with a reduced number of transitions and hence, reduced space required to store transition information. Essentially, this is achieved by replacing certain DFA state transitions by so-called failure transitions. A small additional computational cost is incurred in recognising whether given strings are part of the language. By using a trie (the term used for a DFA graph that is a tree) and failure transitions, Aho and Corasick [1] generalised the so-called KMP algorithm [7] to multi-keyword pattern matching. There are two versions of their algorithm: the so-called optimal one, which we call *aco*, and a failure one, *acf*. *aco* builds a minimal DFA to find all matches of a given keyword set in a text. *acf* builds, in a first step, a trie using all prefixes of words from the set. Each state therefore represents a string which is the prefix of a keyword. Moreover, the string of a state is spelled out by transitions that connect the start state of the trie to that state. In a second step, *aco* then inserts a failure transition from each state of the trie to some other state. To briefly illustrate the nature failure transitions, suppose p is a state in the trie representing the string and keyword *she* and suppose q is another state representing the prefix string *he* of the keyword $hers$. Then a transition from p to q would indicate that *he* is the longest suffix of *she* that matches a prefix of some other keyword. With appropriate further elaboration (details may be found in [1, 11]) the output of *acf* is an FDFA that is language equivalent to the *aco* one It can also be shown that acf is minimal in the following sense. No other FDFA that is language-equivalent to *aco* can have fewer transitions than acf

An algorithm proposed in [8], called the DFA-Homomorphic Algorithm (DHA), constructs from *any* complete DFA a language-equivalent FDFA. As predicted by the theory [2], the resulting FDFA is not necessarily minimal. The abstract version of the algorithm described in [8] involves the construction of a concept lattice as explained in Section 2. The original version of the algorithm leaves a number of decisions as nondeterministic choices. It also strives for minimality by following a "greedy" heuristic in respect of information embedded in the lattice. However, concrete versions of DHA and the effect of this heuristic have not been tested. Here we propose various alternative concrete versions of the algorithm for different deterministic choices.The *acf* FDFAs provide a benchmark for assessing the performance of these concrete variants of DHA. An *aco* DFA is used as input to each of several variants of the DHA and the resulting DHA-FDFAs are compared against the *acf* version.

An alternative approach to constructing FDFAs from an arbitrary DFA has been proposed by Kumar et al [10][5]. Their technique is based on finding the maximal spanning tree [9] of a suitably weighted nondirected graph that reflects the structure of the underlying DFA. Two algorithms were proposed. One is based on a maximal spanning tree, and the other on a redefined maximal spanning tree. Further details about their algorithms may be found in their original publication. The original maximal spanning tree based algorithm was included in our comparative study for its performance assessment using *acf* as the benchmark.

[5] Their research uses different terminology to that given above. They refer to an FDFA as a delayed-input DFA (abbreviated to D^2FA) and failure transitions are called default transitions.

Various other FDFA related research has been conducted for certain limited contexts. See [5] for an overview. A recent example is discussed in [4], where ideas from [8] were used to modify construction of a so-called factor oracle automaton to use failure transitions, saving up to 9% of symbol transitions.

Section 2 provides the formal preliminaries relevant to this research. In Section 3 we introduce the deterministic variants of the DHA that are subsequently benchmarked. Section 4 outlines the investigation's experimental environment, the data generated, the methods of assessment and the results. Section 5 draws conclusions and points to further research work currently underway.

2 Preliminaries

An *alphabet* is a set of symbols, Σ, of size $|\Sigma|$, and Σ^* denotes the set of all sequences over this alphabet, including the empty sequence, denoted by ϵ. A *string* (or word), s, is an element of Σ^* and its length is denoted $|s|$. Note that $|\epsilon| = 0$. The *concatenation* of strings p and q is represented as pq. If $s = pqw$ then q is a *substring* of s, p and pq are *prefixes* of s and q and qw are *suffixes* of s. Moreover, q is a *proper* substring iff $\neg((p = \epsilon) \vee (w = \epsilon))$. Similarly, pq is a proper prefix iff $w \neq \epsilon$ and qw is a proper suffix iff $p \neq \epsilon$.

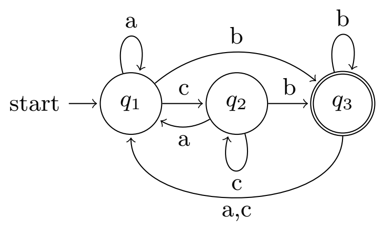
A deterministic finite automata (DFA) is a quintuple, $D = (Q, \Sigma, \delta, F, q_s)$, where Q is a finite set of states; Σ is an alphabet; $\delta \in Q \times \Sigma \not\rightarrow Q$ is the (possibly partial) symbol transition function mapping symbol/state pairs to states; $q_s \in Q$ is the start state; and $F \subseteq Q$ is a set of final states. If δ is a total function, then the DFA is called *complete*. In the case of a complete DFA, the *extension of* δ is defined as $\delta^* \in Q \times \Sigma^* \longrightarrow Q$ where $\delta^*(p, \epsilon) = p$ and if $\delta(p, a) = q$ and $w \in \Sigma^*$, then $\delta^*(p, aw) = \delta^*(q, w)$. A finite string, w, is said to be *accepted* by the DFA iff $\delta^*(q_s, w) \in F$. The language of a DFA is the set of accepted strings.

A failure DFA (FDFA) is a six-tuple, $(Q, \Sigma, \delta, \mathfrak{f}, F, q_s)$, where $D = (Q, \Sigma, \delta, F, q_s)$ is a (not necessarily complete) DFA and $\mathfrak{f} \in Q \not\rightarrow Q$ is a (possibly partial) failure transition function. For all $a \in \Sigma$ and $p \in Q$, the functions δ and $\mathfrak{f}$ are related in the following way: $\mathfrak{f}(p) = q$ for some $q \in Q$ if $\delta(p, a)$ is not defined. The extension of δ in an FDFA context is similar to its DFA version in that $\delta^* \in Q \times \Sigma^* \longrightarrow Q$ and $\delta^*(p, \epsilon) = p$. However:

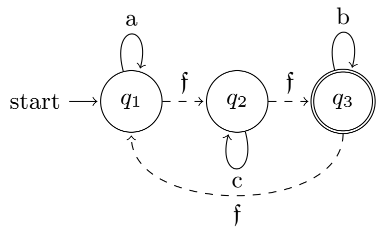
$$\delta^*(p, aw) = \begin{cases} \delta^*(q, w) \text{ if } \delta(p, a) = q \\ \delta^*(q, aw) \text{ if } \delta(p, a) \text{ is not defined and } \mathfrak{f}(p) = q \end{cases}$$

An FDFA is said to accept string $w \in \Sigma^*$ iff $\delta^*(q_s, w) \in F$. An FDFA's language is its set of accepted strings. It can be shown that every complete DFA has a language-equivalent FDFA and *vice-versa.* When constructing an FDFA from a DFA, care must be taken to avoid so-called divergent cycles of failure transitions because they lead to an infinite sequence of failure traversals in string processing algorithms. (Details are provided in [8].)

Subfigure 1a depicts a complete DFA for which $Q = \{q_1, q_2, q_3\}$ and $\Sigma = \{a, b, c\}$. Its start state is q_1 and q_3 is the only final state. Its symbol transitions are depicted as solid arrows between states. Subfigure 1b shows a language-equivalent FDFA where dashed arrows indicate the failure transitions. Note, for example that ab is in the language of both automata. In the DFA case, $\delta^*(q_1, ab) = \delta^*(q_1, b) = \delta^*(q_3, \epsilon)$ In the FDFA case, $\delta^*(q_1, ab) = \delta^*(q_1, b) = \delta^*(q_2, b) = \delta^*(q_3, b) = \delta^*(q_3, \epsilon)$.



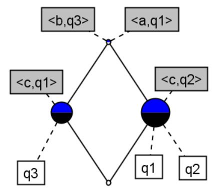
(a) $D = (Q, \Sigma, \delta, q_1, \{q_3\})$



(b) $F = (Q, \Sigma, \delta, \mathfrak{f}, q_1, \{q_3\})$

	$\langle a, q_1\rangle$	$\langle b, q_3\rangle$	$\langle c, q_2\rangle$	$\langle c, q_1\rangle$
q_1	X	X	X	
q_2	X	X	X	
q_3	X	X		X

(c) State/out-transition context



(d) State/out-transition lattice

Fig. 1: Example automata and state/out-transition lattice

This text relies on standard formal concept analysis terminology and definitions. (See, for example [6]). A so-called *state/out-transition* concept lattice can be derived from any DFA. The objects of its formal context are DFA states, $q \in Q$. Each attribute is a pair of the form $\langle a, p\rangle \in \Sigma \times Q$. A state q is deemed to have this attribute if $\delta(q, a) = p$, i.e. if q is the source of a transition on a to p. Subfigure 1c is the state/out-transition context for the DFA in Subfigure 1a and Subfigure 1d is the line diagram of the associated state/out-transition concept lattice. The latter subfigure shows two intermediate concepts, each larger than the bottom concept and smaller than the top, but not commensurate with one another. The right-hand side intermediate concept depicts the fact that states q_1 and q_2 (its extent) are similar in that each as a transition on symbol a to q_1, b to q_3 and on c to q_2 — i.e. the concept's intent is $\{\langle a, q_1\rangle, \langle b, q_3\rangle, \langle c,_{q2}\rangle\}$

Each concept in a state/out-transition lattice can be characterised by a certain value, called its *arc redundancy.* For a concept c it is defined as $ar(c) = (|int(c)| - 1) \times (|ext(c)| - 1)$, where $ext(c)$ and $int(c)$ denote the extent and intent of concept

c respectively. The arc redundancy of a concept represents the number of arcs that may be saved by doing the following:

1. singling out one of the states in the concept's extent;
2. at all the remaining states in the concept's extent, removing all out-transitions mentioned in the concept's intent;
3. inserting a failure arc from each of the states in step 2 to the singled out state in step 1.

The expression, $|ext(c)| - 1$ represents the number of states in step 2 above. At each such state, $|int(c)|$ symbol transitions are removed and a failure arc is inserted. Thus, $|int(c)| - 1$ is the total number of transitions saved at each of $|ext(c)| - 1$ states so that $ar(c)$ is indeed the total number of arcs saved by the above transformation.

The *positive arc redundancy* (PAR) set consists of all concepts whose arc redundancy is greater than zero.

3 The DHA Variants

For the DFA-Homomorphic Algorithm (DHA) to convert a DFA into a language equivalent FDFA, a three stage transformation of the DFA is undertaken. Initially, the DFA is represented as a state/out-transition context. From the derived concept lattices, the PAR set is extracted to serve as input for the DHA.

The basic DHA proposed in [8] is outlined in Algorithm 1. The variable O is used to keep track of states that are not the source of any failure transitions. This is to ensure that a state is never the source of more than one failure transition. Initially all states qualify. A concept c is selected and removed from PAR set, so that c is no longer available in subsequent iterations. The initial version of DHA proposed specifically selecting a concept, c, with maximum arc redundancy. The specification given here leaves open how the choice will be made.

From c's extent, one of the states, t, is chosen to be a failure transition target state. DHA gives no specific criteria for which state in $ext(c)$ to choose. The remaining set of states in $ext(c)$ is denoted by $ext'(c)$. Then, for each state s in $ext'(c)$ that qualifies to be the source of a failure transition (i.e. that is also in O) all transitions in $int(c)$ are removed from s and a failure transition is installed from s to t. Because state s has become a failure transition source state whose target state is t, it may no longer be the source of any other failure transition, and so is removed from O. These steps are repeated until it is no longer possible to install any more failure transitions. It should be noted that in this particular formulation of the abstract algorithm the PAR set is not recomputed to reflect changes in arc redundancy as the DFA is progressively transformed into an FDFA. This does not affect the correctness of the algorithm, but may affect its optimality. Investigating such effects is not within the scope of this study. The third and fifth lines of Algorithm 1, namely

$c := selectAConcept(PAR)$ and
$t := getAnyState(ext(c))$ respectively.

are non-specific in the original formulation of DHA.

Three variants of the algorithm are proposed with respect to the third line. For convenience we represent each variant by a conjunct on the right hand side of an assignment where c is the assignment target. This, of course, is a slight abuse of notation since c is not the outcome of a logical operation, but the selection of a concept from the PAR set according to a criterion represented by $h\_mar(PAR)$, $h\_me(PAR)$ or $h\_mi(PAR)$. Each selection option a different greedy heuristic for choosing concept c from the PAR set. By greedy we mean that an element from the set is selected, based on some maximal or minimal feature, without regard to possible opportunities lost in the forthcoming iterations by making these selections. In addition to these heuristics, a single heuristic is proposed for the fifth line relating to choosing the target state, t, for the failure transitions. These choices are illustrated as colour-coded assignment statements shown in the skeleton Algorithm 2 and are now briefly explained. The rationale for these heuristics will be discussed a section below.

Algorithm 1
```
$O := Q$;   $PAR := \{c \mid ar(c) > 0\}$;
do $((O \neq \emptyset) \wedge (PAR \neq \emptyset)) \rightarrow$
    $c := SelectConcept(PAR)$;
    $PAR := PAR \backslash \{c\}$;
    $t := getAnyState(ext(c))$;
    $ext'(c) := ext(c) \backslash \{t\}$;
    for each $(s \in ext'(c) \cap O) \rightarrow$
        if a failure cycle is not created $\rightarrow$
            for each $((a, r) \in int(c)) \rightarrow$
                $\delta := \delta \setminus \{\langle s, a, r\rangle\}$
            rof;
            $\mathfrak{f}(s) := t$;
            $O := O \backslash \{s\}$ ;
        fi
    rof
od
```

Algorithm 2
```
$O := Q$;   $PAR := \{c \mid ar(c) > 0\}$;
do $((O \neq \emptyset) \wedge (PAR \neq \emptyset)) \rightarrow$
    $c := h\_mar(PAR) \ \vee \ h\_me(PAR) \ \vee \ h\_mi(PAR)$
    $PAR := PAR \backslash \{c\}$
    $t := ClosestToRoot(c)$
    $ext'(c) := ext(c) \backslash \{t\}$;
    for each $(s \in ext'(c) \cap O) \rightarrow$
        if a failure cycle is not created $\rightarrow$
            for each $((a, r) \in int(c)) \rightarrow$
                $\delta := \delta \setminus \{\langle s, a, r\rangle\}$
            rof;
            $\mathfrak{f}(s) := t$;
            $O := O \backslash \{s\}$ ;
        fi
    rof
od
```

The heuristics for choosing concept c from the PAR set in each iteration are as follows: The h_mar heuristic: c is a PAR concept with a *maximum arc redundancy*. The h_mi heuristic: c is a PAR concept with a *maximum intent* size. The h_me heuristic: c is a PAR concept with a *minimum extent* size. Once one of these heuristics has been applied, the so-called *ClosestToRoot* heuristic is used to select a state t in $ext(c)$ to become the target state of failure transitions from each of the remaining states in $ext(c)$. The heuristic means that t is selected as the state in $ext(c)$ that is closest[6] to *aco*'s start state. Transition modifications are subsequently made on the FDFA produced to date, provided that a divergent failure cycle is not produced.

[6] Since a trie has no cycles, the notion of "closest" here simply means a state with the shortest path from the start state to that state.

4 The Experiment

The experiments were conducted on an Intel i5 dual core CPU machine, running Linux Ubuntu 14.4. Code was written in C++ and compiled under the GCC version 4.8.2 compiler.

It can easily be demonstrated that if there are no overlaps between proper prefixes and proper suffixes of keywords in a keyword set, then the associated *acf* FDFA's failure transitions will all loop back to its start state, and out *ClosestToRoot* heuristic will behave similarly. To avoid keyword sets that lead to such trivial *acf* FDFAs, the following keyword set construction algorithm was devised.

Keywords (also referred to as patterns) are from an alphabet set of size 10. Their lengths range from 5 to 60 characters. Keyword sets of sizes $5, 10, 15, \ldots, 100$ respectively are generated. For each of these 20 different set sizes, twelve alternative keyword sets are generated. Thus in total $12 \times 20 = 240$ keyword sets are available.

To construct a keyword set of size N, an initial N random strings are generated[7]. Each such string has random content taken from the alphabet and random length in the range 5 and 30. However, for reasons given below, only a limited number of these N strings, say M, are directly inserted into the keyword set. The set is then incrementally grown to the desired size, N, by repeating the following:

> Select a prefix of random length, say p, from a randomly selected string in the current keyword set. Remove a string, say w, from the set of strings not yet in the keyword set. Insert either pw or wp into the keyword set.

Steps are taken to ensure that there is a reasonable representation of each of these three differently constructed keywords in a given keyword set.

These keyword sets served as input to the SPARE-PARTS toolkit [12] to create the associated *acf* FDFAs and the *aco* DFAs. A routine was written to extract state/out-transition contexts from the *aco* DFAs. These contexts were used by the lattice construction software package known as FCART (version 0.9)[3] supplied to us by National Research University Higher School of Economics (Moscow, Russia). The DHA variants under test used resulting concept lattices to generate the FDFAs. As previously mentioned, a Kumar et al [10] algorithm was also implemented to generate FDFAs from the DFAs. These will be referenced as *kum* FDFAs.

Figures 2 and 3 give several views of the extent to which the DHA-based and *kum* FDFAs correspond with *acf* FDFAs. The experimental data is available online[8]. For notational convenience $\mathfrak{f}^i_{jk}$ denotes the set of failure transitions of

[7] Note that all random selections mentioned use a pseudo-random number generator.
[8] The experimental data files can be found at this URL:
http://www.fastar.org/wiki/index.php?title=Conference_Papers#2015.

the FDFA corresponding to k^{th} keyword set of size $5j$ that was generated by the algorithm variant $i \in FA \backslash \{aco\}$, where $k \in [1, 12]$, $j \in [1, 20]$ and $FA = \{acf, aco, mar, mi, me, kum\}$. Similarly, δ^i_{jk} refers to symbol transition sets of the associated FDFAs and, in this case, also the *aco* DFAs if $i = aco$. A dot notation in the subscript is used for averages. Thus, for some $i \in FA \backslash \{aco\}$ we use $|\mathfrak{f}^i_{j.}|$ to denote the average number of failure transitions in the i-type FDFAs produced by the 12 keyword sets of size $5j$, and similarly $|\delta^i_{j.}|$ represents the average number of symbol transitions.

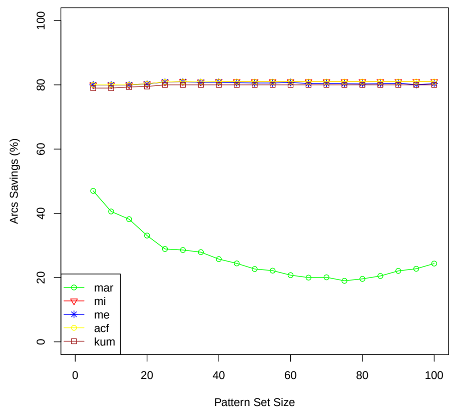
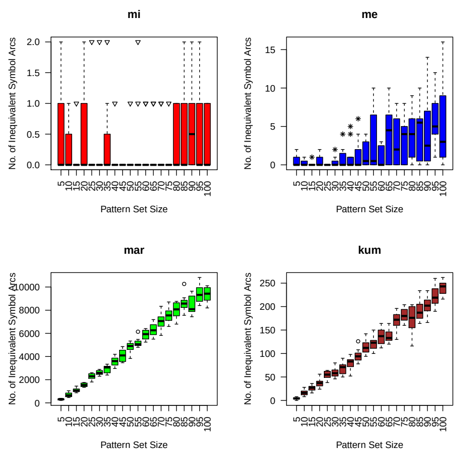


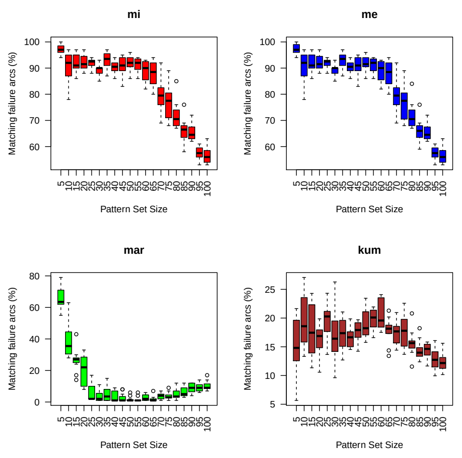
Fig. 2: $\dfrac{|\delta^{aco}_{j.}| - (|\delta^i_{j.}| + |\mathfrak{f}^i_{j.}|)}{|\delta^{aco}_{j.}|} \times 100$

Figure 2 shows how many more transitions *aco* automata require (as a percentage of *aco*) compared to each of the FDFA variants. Note that data has been averaged over the 12 keyword set samples for each of the set sizes and note that the FDFA transitions include both symbol and failure transitions. The minimal *acf* FDFAs attain an average savings of about 80% over all sample sizes and the *mi*, *me* and *kum* FDFAs track this performance almost identically. Although not clearly visible in the graph, the *me* heuristic shows a slight degradation for larger set sizes, while the *kum* FDFAs consistently perform about 1% to 2% worse. By way of contrast, the *mar* heuristic barely achieves a 50% savings for small sample sizes, and drops below a 20% savings for a sample size of about 75, after which there is some evidence that it might improve slightly.

The fact that the percentage transition savings of the various FDFA variants closely correspond to that of *acf* does not mean that the positioning of the failure and symbol transitions should show a one-to-one matching. The extent to which the transitions precisely match one another is shown in Figure 3. These box-and-



(a) $|\delta^{acf}_{jk}| - |\delta^{acf}_{jk} \cap \delta^{i}_{jk}|$



(b) $\dfrac{|\mathfrak{f}^{i}_{jk} \cap \mathfrak{f}^{acf}_{jk}|}{|\mathfrak{f}^{acf}_{jk}|} \times 100$

Fig. 3: Transition Matches

whisker plots show explicitly the median, 25th and 75th percentiles as well as outliers of each of the 12 sample keyword sets of a given size. Subfigure 3a shows the number of symbol transitions in *acf* FDFAs that do not correspond with those in *mi*, *me*, *mar* and *kum* respectively. Subfigure 3b shows the percentage of *acf* failure transitions matching those of the FDFAs generated by *mi*, *me*, *mar* and *kum* respectively.

The symbol transitions for the *mi* heuristic are practically identical to those of *acf*, differing by at most two. Differences are not significantly related to sample size. Differences for *me* are somewhat larger, increasing slightly with larger sample size, though still relatively modest in relation to the overall number of FDFA transitions. (There are $|Q| - 1$ transitions in the underlying trie.) In the cases of *mar* and *kum*, the differences are approximately linearly dependent on the size of the keyword set, reaching over 9000 and 250 respectively for keywords sets of size 100.

The failure transition differences in regard to *mi* and *me* show a very similar pattern as keyword size increases. Only in isolated instances do they fully match those of *acf*, but the matching correspondence drops from a median of more than 95% in the case of the smallest keywords sets to a median of about 50% for the largest keyword sets. The median *kum* failure transition correspondence with *acf* is in a range of about 12-18% for all pattern set sizes. However, in the case of *mar*, the degree of correspondence is much worse: at best the median value is just over 60% for small keyword sets, dropping close to zero for medium range keyword set sizes, and then increasing slightly to about 10% for the largest keyword sets.

Overall, Figures 2 and 3 reveal that there is a variety of ways in which failure transitions may be positioned in an FDFA, and that lead to very good—in many cases even optimal—transition savings. It is interesting to note that even in the *kum* FDFAs, the total number of transition savings is very close to optimal, despite relatively large differences in the positioning of the transitions. However, the figures also show that this flexibility in positioning failure transitions to achieve good arc savings eventually breaks down, as in the case of the *mar* FDFAs.

One of the reasons for differences between *acf* FDFAs and the others is that some implementations of the *acf* algorithm, including the SPARE-PARTS implementation, inserts a failure arc at *every* state (except the start state), even if there is an out-transition on every alphabet symbol from a state. Such a failure arc is of course redundant. Inspection of the data showed that some of the randomly generated keyword sets lead to such "useless" failure transitions, but they are so rare that they do not materially affect the overall observations.

The overall rankings of the output FDFAs of the various algorithms to *acf* could broadly be stated as $mi > me > kum > mar$. This ranking is with respect to closeness of transition placement to *acf*. Since the original focus of this study was to explore heuristics for the DHA algorithm, further comments about the *kum* algorithm are reserved for the next section.

The rationale for the *mar* heuristic is clear: it will cause the maximum savings in transitions in a given iteration. It was in fact the initial criterion proposed in [8]. It is therefore somewhat surprising that it did not perform very well in comparison to other heuristics. It would seem that, in the present context, it is too greedy—i.e. by selecting a concept whose extent contains the set of states that can effect maximal savings such that in one iteration, it deleteriously eliminates from consideration concepts whose extent contains some of those states in subsequent iterations. Note that, being based on the maximum of the product of extent and intent sizes, it will tend to select concepts in the middle of the concept lattice.

When early trials in our data showed up *mar*'s relatively poor performance, the *mi* and *me* heuristics were introduced to prioritise concepts in the top or bottom regions of the lattice. These latter two heuristics will maximize the number of symbol transitions to be removed *per state* when replacing them with failure transitions, in so far as concepts with large intents tend to have small extents and *vice-versa*. Although such a relationship is, of course, data-dependent, random data tends in that direction, as was confirmed by inspection of our data.

These two heuristics appear to be rather successful at attaining *acf*-like FDFAs. However, the *ClosestToRoot* heuristic has also played a part in this success. Note that the *acf* failure transitions are designed to record that a suffix of a state's string is also a prefix of some other state's string. Thus, $f(q) = p$ means that a suffix of state q's string is also a prefix of state p's string. However, since there may be several suffixes of q's string and several states whose prefixes meet this criterion, the definition of f requires that the *longest possible* suffix of q's string should be used. This ensures that there is only one possible state, p, in the trie whose prefix corresponds to that suffix. Thus, on the one hand, *acf* directs a failure transition "backwards" towards a state whose depth is less than that of the current state. On the other hand, *acf* selects a failure transition's target state to be as *far* as possible from the start state, because the suffix (and therefore also the prefix) used must be maximal in length.

The *ClosestToRoot* heuristic approximates the *acf* action in that it also directs failure transitions backwards towards the start state. However, by selecting a failure transition's target state to be as *close* as possible from the start state, it seems to contradict *acf* actions. It is interesting to note in Subfigure 3b that both *mi* and *me* show a rapid and more or less linear decline in failure transition matchings with respect to *acf* when pattern set size reaches about 65. We conjecture that for smaller keyword sizes, the*ClosestToRoot* heuristic does not conflict significantly with *acf*'s actions because there are few failure target states from which to choose. When keyword set sizes become greater, there is likely to be more failure target states from which to choose, and consequently less correspondence between the failure transitions chosen according to differing criteria.This is but one of several matters that has been left for further study.

5 Conclusions and Future Agenda

Our ultimate purpose is to investigate heuristics for building FDFAs from *generalised* complete DFAs—a domain where optimal behaviour is known *a priori* to be computationally hard. The comparison against *acf* FDFAs outlined above is a firm but limited starting point. The next step is to construct complete DFAs from randomly generated FDFAs and examine the extent to which the heuristics tested out in this study can reconstruct the latter from the former. Because generalised DFAs can have cycles, the *ClosestToRoot* heuristic will be generalised by using Dijktra's algorithm for calculating the shortest distance from the start state to each DFA state. It remains to be seen whether *mar* will perform any better in the generalised context.

The relatively small alphabet size of 10 was dictated by unavoidable growth in the size of the associated concept lattices. Even though suitable strategies for trimming the lattice (for example by not generating concepts with arc redundancy less than 2) are being investigated, it is recognised that use of DHA will always be constrained by the potential for the associated lattice to grow exponentially. Nevertheless, from a theoretical perspective a lattice-based DHA approach to FDFA generation is attractive because it encapsulates the solution space in which a minimal FDFA might be found—i.e. each ordering of its concepts maps to a possible language-equivalent FDFA that can be derived from DFA and at least one such ordering will be a minimal FDFA.

The *kum* FDFA generation approach is not as constrained by space limitations as the DHA approach and in the present experiments it has performed reasonably well. In the original publication, a somewhat more refined version is reported that attempts to avoid unnecessary chains of failure transitions. Future research should examine the minimising potential of this refined version using generalised DFAs as input and should explore more fully the relationship between these *kum*-based algorithms and the DHA algorithms.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.
2. H. Björklund, J. Björklund, and N. Zechner. Compression of finite-state automata through failure transitions. *Theor. Comput. Sci.*, 557:87–100, 2014.
3. A. Buzmakov and A. Neznanov. Practical computing with pattern structures in FCART environment. In *Proceedings of the International Workshop "What can FCA do for Artificial Intelligence?" (FCA4AI at IJCAI 2013), Beijing, China, August 5, 2013.*, pages 49–56, 2013.
4. L. Cleophas, D. G. Kourie, and B. W. Watson. Weak factor automata: Comparing (failure) oracles and storacles. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2013*, pages 176–190, Czech Technical University in Prague, Czech Republic, 2013.
5. M. Crochemore and C. Hancart. Automata for matching patterns. In S. A. Rozenberg G., editor, *Handbook of Formal Languages*, volume 2, Linear Modeling: Background and Application, pages 399–462. Springer-Verlag, 1997. incollection.
6. B. Ganter, G. Stumme, and R. Wille, editors. *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer, 2005.
7. D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.
8. D. G. Kourie, B. W. Watson, L. Cleophas, and F. Venter. Failure deterministic finite automata. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2012*, pages 28–41, Czech Technical University in Prague, Czech Republic, 2012.
9. J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.
10. S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. S. Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pisa, Italy, September 11-15, 2006*, pages 339–350, 2006.
11. B. W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms.* PhD thesis, Eindhoven University of Technology, Sept. 1995.
12. B. W. Watson and L. G. Cleophas. SPARE Parts: a C++ toolkit for string pattern recognition. *Softw., Pract. Exper.*, 34(7):697–710, 2004.