DCFS 2010 (Draft)

Descriptional Complexity of Formal Systems
Deadline for submissions: April 25, 2010
Final versions: July 8, 2010

State Elimination Ordering Strategies: Some Experimental Results

Nelma Moreira Davide Nabais[(A)] Rogério Reis

DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal
nam@ncc.up.pt dnabais@ncc.up.pt rvr@ncc.up.pt

Abstract. Recently, the problem of obtaining a short regular expression equivalent to a given finite automaton has been intensively investigated. Algorithms for converting finite automata to regular expressions have an exponential blow-up in the worst-case. To overcome this, simple heuristic methods have been proposed. In this paper we analyse some of the heuristics presented in the literature and propose new ones. We also present some experimental comparative results based on uniform random generated deterministic finite automata.

Keywords: finite automata, regular expressions, state elimination method, heuristics

1 Introduction

Recently, the problem of obtaining a short regular expression equivalent to a given finite automaton has been intensively investigated. An extensive survey was presented by Ellul *et al.* [EKSW05], and more recently by Gruber and Holzer [GH08b]. It is well known that the problem of obtaining a minimal regular expression is PSPACE-complete and NP-complete for acyclic automata [JR93]. It is also inefficient to approximate a minimal regular expression [GS07], unless P=PSPACE. Classic algorithms for converting finite automata to regular expressions can produce regular expressions of size $\mathcal{O}(nk4^n)$ in the worst case, where n is the number of states and k the alphabet size of the correspondent automaton. Several exponential lower bounds are provided in the literature [EKSW05, GH08a] showing that the exponential blow-up is unavoidable. For specific classes of automata, better upper bounds can be found [EKSW05, GF08, Sak05, MR09]. In particular, Gruber and Holzer [GH08b] presented an algorithm that converts an n-state deterministic finite automaton (DFA) over a binary alphabet into a regular expression of size at most $\mathcal{O}(1.742^n)$. In general, to obtain shorter regular expressions it is essential the order in which the automaton's states are considered in the conversion. To tackle the problem

Research partially funded by Fundação para a Ciência e Tecnologia (FCT) and Program POSI, and by projects ASA (PTDC/MAT/65481/2006) and CANTE (PTDC/EIA-CCO/101904/2008).

[(A)] Davide Nabais is funded by a LIACC-FCT scholarship for young undergraduate researchers.

of obtaining an optimal ordering in a feasible manner, heuristic methods have been proposed [DM04, HW07, AH09].

In this paper we analyse some of the heuristics presented in the literature and propose new ones. To test their performance, some experimental results were carried out using statistically significant samples obtained with an uniform random generator. The paper is organized as follows. In the next section some basic notions are reviewed. Section 3 summarizes the conversions from finite automata to regular expressions, and in particular the state elimination method. Section 4 describes some elimination ordering strategies and two new ones are proposed. In Section 5 experimental results are analysed and Section 6 concludes.

2 Preliminaries

We recall some basic notions of digraphs, finite automata and regular expressions. For more details we refer the reader to standard books [HMU00, Sak09, Har69].

A digraph $D = (V, E)$ consists of a finite set V of vertices and a set E of ordered pairs of vertices, called *arcs*. If (u, v) in E, u is *adjacent to* (or incident to) v and v is *adjacent from* u. For each vertex v, the *indegree* of v is the number n_i of vertices adjacent to it and the *outdegree* of v is the number n_o of vertices adjacent from it, and we write $v(n_i; n_o)$. An arc (u, v) can be denoted by uv. A *path* between v_0 and v_n is a sequence $v_0v_1, v_1v_2, \ldots, v_{n-1}v_n$ of arcs, and is denoted by $\overline{v_0 \cdots v_n}$, or $\overline{v_0 \cdots v_k \cdots v_n}$, for $1 \leq k < n$. A path is *simple* if all the vertices in it are distinct. The length of a path is the number of arcs in the path. A path is a *cycle* if $v_0 = v_n$ and $n \geq 1$. A digraph that has no cycles is called *acyclic*.

We now review some notions and notation from formal languages and finite automata. Let Σ be a finite alphabet and $\Sigma^\star$ be the set of words over Σ. The empty word is denoted by ε. A *language* over Σ is a subset of $\Sigma^\star$. A *regular expression* (r.e.) α over Σ represents a regular language $\mathcal{L}(\alpha) \subseteq \Sigma^\star$ and is inductively defined by: $\emptyset$ is a r.e. and $\mathcal{L}(\emptyset) = \emptyset$; ε is a r.e. and $\mathcal{L}(\varepsilon) = \{\varepsilon\}$; $a \in \Sigma$ is a r.e. and $\mathcal{L}(a) = \{a\}$; if α_1 and α_2 are r.e., $(\alpha_1 + \alpha_2)$, $(\alpha_1\alpha_2)$, and $(\alpha_1)^\star$ are r.e., respectively with $\mathcal{L}((\alpha_1 + \alpha_2)) = \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$, $\mathcal{L}((\alpha_1\alpha_2)) = \mathcal{L}(\alpha_1)\mathcal{L}(\alpha_2)$, and $\mathcal{L}((\alpha_1)^\star) = \mathcal{L}(\alpha_1)^\star$. The *alphabetic size* of an r.e. α is the number of alphabetic symbols of α and is denoted by $|\alpha|_\Sigma$. Let R be the set of regular expressions over Σ. Two regular expressions α and β are *equivalent* if $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$, and we write $\alpha = \beta$. With this interpretation, the algebraic structure $(\mathsf{R}, +, \cdot, \emptyset, \varepsilon)$ constitutes an idempotent semiring, and with the unary operator $\star$, a Kleene algebra. Using these algebraic properties as (simplification) rewrite rules, it is possible to decide if two regular expressions are equivalent, but no algorithm is known to minimize a given regular expression (except a `brute-force` one).

A *non-deterministic finite automaton* (NFA) $\mathcal{A}$ is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is the alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, q_0 the initial state and $F \subseteq Q$ is the set of final states. For $q \in Q$ and $a \in \Sigma$, we denote the set $\{p \in Q \mid (q, a, p) \in \delta\}$ by $\delta(q, a)$, and we can extend this notation to $w \in \Sigma^\star$, and to $R \subseteq Q$. The *language* recognized by

$\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\star \mid \delta(q_0, w) \cap F \neq \emptyset\}$. An NFA is *deterministic* (DFA) if for each pair $(q, a) \in Q \times \Sigma$, $|\delta(q, a)| \leq 1$. A DFA is complete if δ is a total function. An NFA is *initially-connected* if for each state $q \in Q$ there exists a word $w \in \Sigma^\star$ such that $q \in \delta(q_0, w)$. A complete initially-connected DFA is denoted by ICDFA. An NFA is *trim* if it is initially-connected and if every state is *useful*, *i.e.*, for all $q \in Q$ there exist a word $w \in \Sigma^\star$ such that $F \cap \delta(q, w) \neq \emptyset$. The *underlying digraph* of an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is the digraph $D = (Q, E)$ such that $E = \{(q, q') \mid q, q' \in Q$ and $\exists a \in \Sigma \cup \{\varepsilon\}$ such that $(q, a, q') \in \delta\}$. Note that even if there can be more than one symbol of Σ between two states q and q', only one arc exists in the underlying digraph.

For the conversion from NFAs to r.e.'s extended finite automata are considered. An *extended finite automaton* (EFA) $\mathcal{A}$ is a quintuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0 and F are as before, and $\delta : Q \times Q \to \mathsf{R}$. We assume that $\delta(q, q') = \emptyset$, if the transition from q to q' is not defined. Any NFA can be easily transformed into an equivalent EFA, with the same underlying digraph: for each pair of states (q, q') one needs to construct a regular expression $a_1 + \cdots + a_n$ such that $(q, a_i, q') \in \delta$, $a_i \in \Sigma \cup \{\varepsilon\}$, $1 \leq i \leq n$. This transformation corresponds to eliminate *parallel* transitions. Whenever appropriated we will use the same terminology both for digraphs and for automata.

3 From Finite Automata to Regular Expressions

Kleene's theorem [Kle56] establishing the equivalence between languages accepted by finite automata and represented by regular expressions provided proof that a language accepted by an NFA can be represented by a r.e.. McNaughton and Yamada [MY60] presented a recursive algorithm that calculates a r.e. from an NFA based on the computation of the transitive closure of the underlying digraph. Brzozowski and McCluskey [BJ63] introduced a method now known as *state elimination algorithm* (SEA) that considers EFAs and leads, in general, to simpler computations and shorter r.e.'s. A third method exists based on solving a system of linear equations akin a Gaussian elimination process [Ard60, Koz94]. This last approach is interesting as linear algebra or optimization techniques can be adapted in order to provide new methods to obtain r.e.'s. Sakarovitch [Sak05, Sak09] studied the relationship between the three methods and in particular showed that given an order in the set of states Q the regular expressions obtained by two different methods can be reduced to each other by the application of a specific subset of algebraic properties.

Most improvements and heuristic methods are based on the state elimination method and try to identify state orderings that lead to shorter r.e.'s.

3.1 State Elimination Method Revisited

The state elimination algorithm takes as input an EFA and produces an equivalent r.e.. In each step, a non-initial and non-final state of the EFA is eliminated (deleted) and the transitions are changed in such way that the new and the older EFAs are equivalent. Usually it is assumed that the input EFA is trim and *normalized*, *i.e.*, the initial state has no incoming transitions, there is only a final state and that state has no outgoing transitions. Every EFA (or NFA) can be transformed into an equivalent normalized EFA. Formally, let $\mathcal{A} = (Q, \Sigma, \delta, q_o, F)$ be an EFA, then:

Normalization:

(NI) If there is $q \in Q$ such that $\delta(q, q_0) \neq \emptyset$, then add a new state i to Q, define $\delta(i, q_0) = \varepsilon$, and set i as the new initial state.

(NII) If $|F| > 1$ or exists $q \in F$ and $q' \in Q$ such that $\delta(q, q') \neq \emptyset$, then add a new state f to Q and a transition $\delta(q, f) = \varepsilon$, for all $q \in F$. The set of final states becomes $\{f\}$.

Without lost of generality, let $A' = (Q', \Sigma, \delta', i, f)$ denote the new normalized EFA. Let $\alpha_{qq'}$ denote the regular expression $\delta(q, q')$. Normalization is preserved when the below *state elimination* process is performed.

State Elimination:

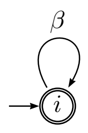
(EI) If $Q = \{i, f\}$, then the resulting regular expression is α_{if}, and the algorithm terminates. Otherwise continue to step (EII).

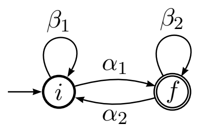
(EII) **Choose** $q \in Q \setminus \{i, f\}$. Eliminate q from A', considering $Q' \setminus \{q\}$ the new set of states, and for each $q_1, q_2 \in Q' \setminus \{q\}$,

$$\delta'(q_1, q_2) = \alpha_{q_1 q_2} + \alpha_{q_1 q}\alpha_{qq}^{\star}\alpha_{qq_2},$$

Continue to step (EI).

Hopcroft *et al.* [HMU00] presented a slight variation of the above algorithm that omits the normalization step. Considering that there is only one final state, state elimination ends with one of the following EFAs (where some r.e.'s can be $\emptyset$):





Initial state is final. There are two different states.

In the left case, the final regular expression is $\beta^{\star}$ and in the right case, the final regular expression can be $\beta_1^{\star}\alpha_1(\beta_2 + \alpha_2\beta_1^{\star}\alpha_1)^{\star}$ or any shorter r.e. if some of the transitions are labelled by $\emptyset$. When $|F| > 1$ the normalization step (NII) should be considered. We refer, by abuse of language, to this algorithm as the SEA *without normalization* (SEAwn). It has the advantage of avoiding unnecessary ε transitions, and, as we will see in Section 4.3, it exhibits a better performance for the elimination strategies.

4 State Elimination Orderings

The importance of the order in which the states are considered in the conversion, was noticed by the authors of the early algorithms. McNaugthon and Yamada suggested that states with higher in- and outdegrees should be considered at the end. Brzozowski and McCluskey proposed to eliminate first the states $q \in Q$ such that $q(1;1)$, *i.e.*, q connects two other states in *series*:

$$\begin{tikzcd} q' \arrow[r, "\alpha"] & q \arrow[r, "\beta"] & q'' \end{tikzcd}$$

Acyclic NFAs for which in each step of the state elimination process there is a state satisfying these conditions were studied by Moreira and Reis [MR09] and called SP-automata. For this class it is possible to obtain a linear size r.e. in $\mathcal{O}(n^2 \log n)$ time. If an acyclic NFA is not SP, it must be reduced by series-parallel elimination to one that contains a subgraph of the form:

$$\begin{tikzcd} & s_2 \arrow[ddr, "c"] \arrow[rr, "d"] & & \\ s_1 \arrow[ur, "a"] \arrow[drr, "b"'] & & & s_4 \\ & & s_3 \arrow[ur, "e"'] & \end{tikzcd}$$

And, in general, it is not easy to see which elimination ordering should be considered.

The SP-automata strategy was extended by Gulan and Fernau [GF08] for a specific case of cyclic NFAs. SP-automata belong to the class of graphs which excludes a complete graph as a minor. For this class, Ellul and *et al.* proved that there are r.e.'s which size is less than $e^{\mathcal{O}(\sqrt{n})}$. Gruber and Holzer extended this work to DFAs, providing an algorithm with a guaranteed performance of $\mathcal{O}(1.742^n)$ for binary alphabets.

4.1 Delgado and Morais Heuristics

In each step of the state elimination process, given $q(m;l)$, the contribution of this state for the size of the final regular expression can be measured by

$$W(q) = (l-1)\sum_{i=1}^{m} |\alpha_{q_i q}| + (m-1)\sum_{j=1}^{l} |\alpha_{q q_j}| + (ml-1)|\alpha_{qq}|. \qquad (1)$$

Delgado and Morais [DM04] proposed a strategy (DM) that in each step eliminates a state q with the lowest *weight* $W(q)$. Although this heuristic is quite simple and runs in $\mathcal{O}(n^2)$, the experimental results provides evidence that it has very good performance. Recently, Gruber *et al.* [GHT09] presented more experimental results which showed statistical significance and were based on uniform random generated ICDFAs, where this heuristic almost always outperforms several others. Our results corroborate this good performance. In particular, when applied to an SP-automaton, this heuristics always selects a state q such that $q(1;1)$, producing a linear size r.e..

4.2 Han and Wood Heuristics

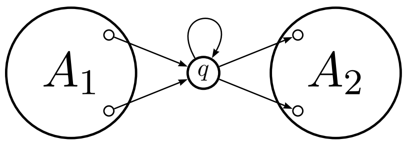
Han and Wood [HW07] introduced the notion of *bridge state* which leads to a decomposition of the EFA, therefore of the elimination process. That notion was redefined by Ahn and Han [AH09], as follows: a state q is a *bridge state* if it satisfies the following conditions:

(BI) q is neither initial nor final;

(BII) For any $f \in F$, each path $\overline{i \cdots f}$ must pass through q, *i.e.*, must be of the form $\overline{i \cdots q \cdots f}$, where i is the initial state;

(BIII) q does not participate in any cycle except for a loop.

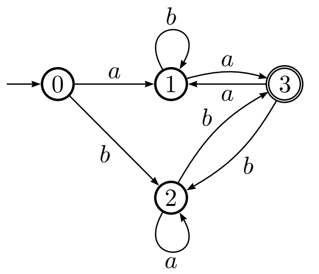
Note that bridge states correspond to the usual notion of *cut points*, with the extra constraint (BIII). Bridge states can be found in linear time, and it was proved that in an optimal elimination ordering the bridge states must be the last ones. This is easy to see because the automaton can be *decomposed* into two subautomata $\mathcal{A}_1$ and $\mathcal{A}_2$, such that a bridge state q corresponds to the final state of $\mathcal{A}_1$ and the initial state of $\mathcal{A}_2$:



Ahn and Han present some empirical results of this strategy (that we designed by HW) combined with the one based on state weights (DM) and also with one that performs a *parallel* decomposition of the EFA. Although the dataset used is random generated, it is not uniform nor statistically significant.

4.3 SEA Without Normalization

Consider the following simple DFA:

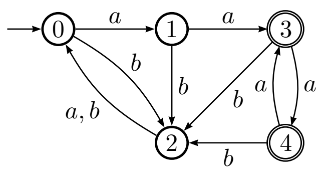


Applying the SEA with normalization to this DFA and using the DM strategy, the first state to be eliminate corresponds to the initial state (*i.e.* it is the one with small weight). This will lead to a r.e. with the highest alphabetic size (29), within all that can be obtained by state elimination. The elimination ordering is 0, 3, 1, 2.

On the other hand, if we consider a SEA with the Hopcroft *et al.* approach (such that the initial state is only considered at the end) applying the DM strategy will lead to a r.e. with the smallest alphabetic size (12). Now, the elimination ordering is 2, 1 (as the two other states are fixed). This strategy corresponds to combine the DM strategy with one where the initial state is the last to be eliminated. Our experimental results below show that this approach (SEAwn) improves, in general, the strategies we considered.

4.4 A New Heuristic: Counting Cycles

Consider, now, the following DFA



The DM heuristics produces a r.e. with alphabetic size 29 or 26, if either SEA or SEAwn is considered. The corresponding elimination order are 1, 4, 0, 2, 3 and 1, 3, 2, 4, respectively. For this DFA the optimal alphabetic size for r.e. obtained by the state elimination method is 16 (and the worst is 126). Instead of the weight of a state being the weighted summation of its in- and out-degrees, one can consider the number of cycles that pass through it (multiplicities included). In this particular case the obtained r.e. has size 19. The number of cycles for each state is, by increasing identifier order, 4, 3, 4, 3 and 2, respectively.

Two strategies can be developed to obtain an elimination ordering:

(CI) statically determine the number of cycles for each state q, of the original automaton (CS); this can be achieved in $\mathcal{O}(n^2)$.

(CII) dynamically determine those values after each elimination step (CD); this can be achieved in $\mathcal{O}(n^3)$.

In the second case, (CII), instead of the multiplicities, the alphabetic size of each transition label is considered.

5 Experimental Results

Each of the state elimination algorithms described before was implemented in Python within the FAdo system [MR05, AAA$^+$09, FAd10]. The experiments were undertaken with samples of 10,000 uniform random generated ICDFAs [AMR07] with a fixed number of states (n) and alphabet size (k). The sample size ensures the statistical significance with a 95% confidence level within a 1% error margin. Most of the tests were performed for automata with $n \in \{10, 20, 50\}$ states and $k \in \{2, 3, 5, 10, 26, 100\}$ symbols. Each generated automaton is represented by a *canonical* string. Assuming an ordering on the alphabet, the states are numbered from 0 to $n - 1$, 0 being the initial state. The string representation is a list of states reached from each state by increasing order of symbols and of state numbering, beginning with the initial state. For example, the string for the DFA of Section 4.3, considering $a < b$, is 12312312.

Experiments were carried out considering the following goals:

- to determine the density of occurrence of bridge states in (complete) DFAs.
- to test the performance of SEAwn, *i.e.* the state elimination method without normalization, independently of other elimination ordering strategies;
- to test the performance of the strategies based on counting the number of cycles.

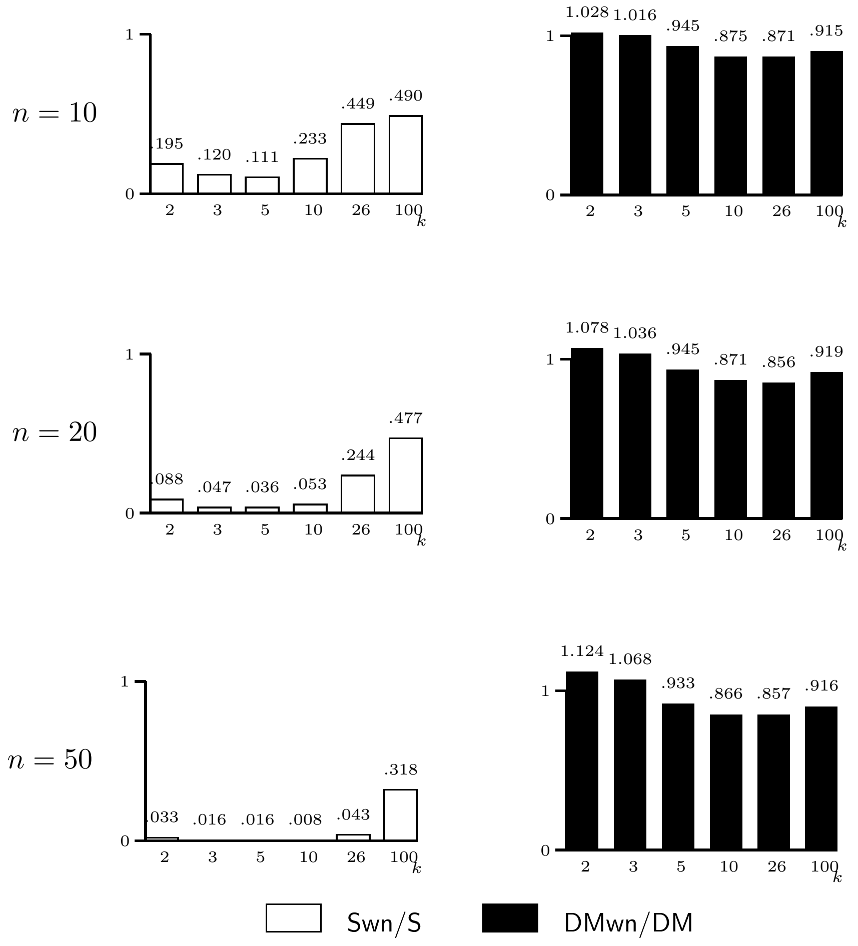
5.1 Bridge States Density

The performance of the strategy HW proposed by Han and Wood, and described in Section 4.2, heavily depends on the existence of bridge states in a finite automaton. We estimated the occurrence of these states in ICDFAs, and their average position in the ICDFA canonical string. In the string representation, an early position corresponds to a closer proximity to the initial state. Thus this index measures the state distance from the initial state and gives information about the number of states of each subautomaton in which the ICDFA can be decomposed. In the following table, and for each sample, *tot* is the total number of bridge states, *num* is the number of ICDFAs with at least a bridge state and *pos* is their average position in the ICDFA canonical string. The table values suggest that bridge states are very rare and a bridge state is usually the initial state or adjacent from it. Note that for larger alphabets ($k \geq 10$) no bridge states, at all, were found.

	$k = 2$			$k = 3$			$k = 5$			$k = 10$		
	tot	num	pos	tot	num	pos	tot	num	pos	tot	num	pos
$n = 10$	3252	2327	0.824	829	707	0.458	88	82	0.193	0	0	N/A
$n = 20$	3506	2375	1.224	757	634	0.486	73	71	0.123	0	0	N/A
$n = 50$	3499	2411	1.375	758	649	0.451	69	63	0.115	0	0	N/A

5.2 SEAwn Performance

To test the performance of the SEAwn method, several elimination ordering strategies were considered. A trivial order is the one in which the states occur in the ICDFA canonical string. This ordering produces very bad results (even compared with a random one) but here we wanted to test the effect of the prior automata normalization. The correspondent algorithms are S and Swn, respectively. We also considered the DM strategy with the SEAwn method (DMwn). For each pair of algorithms, the ratio between the average r.e. alphabetic sizes was computed. The following bar charts summarize some of the results. The Swn method (without normalization) always outperforms the S (with normalization). Because the r.e. sizes are huge some ratios are very small. For example, a ratio of 0.08, for $n = 50$ and $k = 10$, corresponds to the diminishing of two orders of magnitude (from 10^{27} to 10^{25}). The DMwn method can achieve an improvement of 15% over the DM one.



5.3 Cycle Heuristic Performance

The two heuristics presented in Section 4.4, CS and CD, were implemented using the SEAwn method. It was then natural to compare their performance with DMwn, the best heuristic so far. The following table summarizes the results. The third to the fifth columns have the average r.e. alphabetic sizes obtained for each of the mentioned heuristics. The sixth column corresponds to the average of the minimum value of the three, *the best of the 3* (B3). The three last columns contain the maximum values obtained by each of the heuristics.

k	n	DMwn	CS	CD	B3	MDMwn	MCS	MCD
2	10	149	144	143	135	864	1014	909
	20	1557	1531	1617	1331	12494	18235	16230
	50	3.5×10^5	4.9×10^5	5.5×10^5	2.5×10^5	7.8×10^6	1.9×10^7	2.5×10^7
3	10	633	617	628	564	4792	4206	5095
	20	23431	25817	27560	18739	339595	365533	428164
	50	2.5×10^8	7.6×10^8	6.5×10^8	1.6×10^8	1.0×10^{10}	1.6×10^{11}	8.9×10^{10}
5	10	4492	4646	4713	3942	32780	34044	35508
	20	1.0×10^6	1.5×10^6	1.4×10^6	8.2×10^5	1.2×10^7	2.8×10^7	2.7×10^7
	50	5.5×10^{12}	3.5×10^{13}	2.0×10^{13}	3.2×10^{12}	4.4×10^{14}	5.3×10^{15}	3.1×10^{15}
10	10	52943	59921	57138	47564	232338	430391	262446
	20	1.8×10^8	3.1×10^8	2.7×10^8	1.4×10^8	1.7×10^9	9.9×10^9	3.2×10^9
26	10	6.0×10^5	7.1×10^5	6.5×10^5	5.8×10^5	1.1×10^6	1.7×10^6	1.5×10^6
	20	3.3×10^{10}	5.7×10^{10}	4.4×10^{10}	2.9×10^{10}	1.3×10^{11}	3.8×10^{11}	1.8×10^{11}
100	10	4.1×10^6	4.2×10^6	4.1×10^6	4.1×10^6	5.3×10^6	5.6×10^6	5.5×10^6
	20	1.5×10^{12}	1.7×10^{12}	1.6×10^{12}	1.4×10^{12}	2.1×10^{12}	2.9×10^{12}	2.7×10^{12}

On average, the heuristics DMwn outperforms the other two, although not always. However, the performance of the cycle heuristics are of the same order of magnitude. The comparison between CS and CD is hard to interpret. The overhead of reevaluate the cycle weights after each step seems not worthwhile. This suggest that the CS strategy is a good choice, even compared with DMwn, as the weights are computed only once. The most important result is that considering the three heuristics a better value is always obtained (B3). This means that when DMwn produces a bad value one of the other two produces a better value, and vice versa. This is surprising, and deserves future research.

6 Conclusions

Several state elimination ordering strategies were analysed and new ones were proposed. Experimental results were conducted with statistical accurate samples of uniform random generated deterministic finite automata. In this context the following conclusions can be drawn:

- a general improvement in all strategies is obtained if the SEA *without normalization* is considered;
- bridge states are very rare;
- the HW strategy clearly clash with the new strategies based on the number of cycles count (CS and CD), because bridge states are cycle free; but, as we saw, their rarity makes this contradiction unimportant;
- the new proposed strategies (CS and CD) are comparable with the DM heuristic; however these new heuristics only outperform, on average, the DM heuristic for automata with small alphabets and small number of states;
- if one takes as strategy, for each automaton, the best result from these three heuristics (DM, CS and CD) a gain of 25% is obtained, with the

same worst case complexity, $\mathcal{O}(n^3)$.

Part of our planned future work is to gain some theoretical understanding of these facts. Furthermore, we conjecture that a more sophisticated hybridization of these three heuristics could lead to even better results.

7 Acknowledgements

We thank the anonymous referees for the many suggested improvements of the paper.

References

[AAA+09] A. Almeida, M. Almeida, J. Alves, N. Moreira, and R. Reis. FAdo and GUItar: tools for automata manipulation and visualization. In S. Maneth, editor, *CIAA 2009: 14th International Conference on Implementation and Application of Automata*, volume 5642 of *LNCS*, pages 65–74, Sidney, July 2009. Springer.

[AH09] J.-H. Ahn and Y.-S. Han. Implementation of state elimination using heuristics. In S. Maneth, editor, *CIAA 2009, 14th International Conference on Implementation and Application of Automata*, volume 5642 of *LNCS*, pages 178–187, Sidney, July 2009. Springer.

[AMR07] M. Almeida, N. Moreira, and R. Reis. Enumeration and generation with a string automata representation. *Theoret. Comput. Sci.*, 387(2):93–102, 2007. Special issue ”Selected papers of DCFS 2006”.

[Ard60] D. N. Arden. Delayed logic and finite state machines. In *Theory of Computing Machine Design*, pages 1–35. U. of Michigan Press, Ann Arbor, 1960.

[BJ63] J. A. Brzozowski and E. J. McCluskey Jr. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. on Electronic Computers*, EC-12(2):67–76, 1963.

[DM04] M. Delgado and J. Morais. Approximation to the smallest regular expression for a given regular language. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *CIAA 2004, 9th International Conference on Implementation and Application of Automata*, volume 3317 of *LNCS*, pages 312–314. Springer, 2004.

[EKSW05] K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: New results and open problems. *J. Aut., Lang. and Combin.*, 10(4):407–437, 2005.

[FAd10] Project FAdo. FAdo: tools for formal languages manipulation. `http://www.ncc.up.pt/FAdo`, Access date:1.1.2010.

[GF08] S. Gulan and H. Fernau. Local elimination-strategies in automata for shorter regular expressions. In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *SOFSEM 2008, Nový Smokovec, Slovakia, 2008, Volume II - Student Research Forum*, pages 46–57, 2008.

[GH08a] H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. In L. Aceto, I. Damgård, L. A. Goldberg, M. MM. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, 35th*

International Colloquium on utomata, Languages and Programming, Part II, volume 5126 of *LNCS*, pages 39–50, Reykjavik, Island, July 2008. Springer.

[GH08b] H. Gruber and M. Holzer. Provably shorter regular expressions from deterministic finite automata. In M. Ito and M. Toyama, editors, *Proceedings of the 12th International Conference Developments in Language Theory*, number 5257 in LNCS, pages 383–395, Kyoto, September 2008. Springer.

[GHT09] H. Gruber, M. Holzer, and M. Tautschnig. Short regular expressions from finite automata: Empirical results. In S. Maneth, editor, *CIAA 2009, 14th International Conference on Implementation and Application of Automata*, volume 5642 of *LNCS*, pages 188–197, Sidney, July 2009. Springer.

[GS07] G. Gramlich and G. Schnitger. Minimizing nfa's and regular expressions. *J. Comput. Syst. Sci.*, 73(6):908–923, 2007.

[Har69] F. Harary. *Graph Theory.* Addison Wesley, 6th edition, 1969.

[HMU00] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2000.

[HW07] Y.-S. Han and D. Wood. Obtaining shorter regular expressions from finite-state automata. *Theoret. Comput. Sci.*, 370:110–120, 2007.

[JR93] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal of Computation*, pages 1117–1141, 1993.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[Koz94] D. C. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, May 1994.

[MR05] N. Moreira and R. Reis. Interactive manipulation of regular objects with FAdo. In *Proceedings of 2005 Innovation and Technology in Computer Science Education (ITiCSE 2005)*, pages 335–339. ACM, 2005.

[MR09] N. Moreira and R. Reis. Series-parallel automata and short regular expressions. *Fundam. Inform.*, 91(3-4):611–629, 2009.

[MY60] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Trans. on Electronic Computers*, EC-9(1):39–47, 1960.

[Sak05] J. Sakarovitch. The language, the expression, and the (small) automaton. In I. Litovshy J. Farré and S. Schmitz, editors, *CIAA 2005, 10th International Conference on Implementation and Application of Automata*, volume 3845 of *LNCS*, pages 15–30. Springer, 2005.

[Sak09] J. Sakarovitch. *Elements of Automata Theory.* CUP, 2009.