Expected Compression Ratio for DFCA: experimental average case analysis

Cezar Câmpeanu, Nelma Moreira, Rogério Reis

Technical Report Series: DCC-2011-07
Version 1.0 December 2011

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL
Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

Expected Compression Ratio for DFCA: experimental average case analysis

Cezar Câmpeanu[1], Nelma Moreira[2], Rogério Reis[2]
ccampeanu@upei.ca, {nam,rvr}@dcc.fc.up.pt

[1] Dept. of Computer Science and Information Technology, University of Prince Edward Island, Charlottetown, P.E.I, Canada C1A 4P3
[2] CMUP, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal

Abstract. In this paper we investigate from a statistical point of view the expected compression ratio between the size of a minimal Deterministic Finite Cover Automaton (DFCA) and the size of the minimal corresponding Determinitic Finite Automaton (DFA). Using sound statistical methods, we extend the experimental study done in [16], thus obtaining a much better picture of the compression power of DFCAs. We compute the expected ratio for the family of all finite languages, but also for various subfamilies of finite languages, such as prefix, suffix-free languages prefix and suffix closed languages, or (un)balanced languages. We also give an example of a family for which the expected compression ratio is very high.

1 Introduction

Since the first publication on cover automata [4], there have been many questions about the compression ratio that these devices are able to produce, compared to minimal Deterministic Finite Automata (DFA). In [16], this subject has been studied and we have examples of families of languages producing an amazing compression, but also languages that have almost no compression.

In this study we give a better insight of the problem, analyzing the whole set of finite languages over a binary alphabet. However, we expect that the same results are valid for any given alphabet with more than two letters. First we give two enumerations of finite languages over the binary alphabet. Because of the combinatorial nature of these sets it is impractical to perform tests over all its elements. That being we provide a uniform random generator for finite languages parametrized by the length of the longest word they accept. Also, we adapt the uniform random generator for various families of finite languages. Thus, we obtain the expected compression ratio of minimal deterministic cover automata (DFCA) to minimal DFA for various subclasses of finite languages, such as:

1. prefix -free,
2. suffix-free,
3. prefix-closed,
4. suffix-closed, and
5. balanced and unbalanced languages.

The paper is organized as follows: we start by giving some notations, we continue introducing two enumerations of finite languages, then we describe uniform random generators for finite languages using tries. Section 5 presents the experimental results obtained.

In Section 6, we present a class of finite languages having a compression ratio greater or equal to a given natural number. We prove that for every language is this family of finite languages we have a compression ratio bigger than a given constant, and at the end of the paper, we summarize our conclusions in Section 7.

2 Preliminaries

The finite languages used in applications are generally very large, and their representation by deterministic finite automata (DFA) need thousands or even millions of states. In [3,4], deterministic finite cover automata (DFCA) are introduced as an alternative representation of finite languages.

We assume the reader to be familiar with the basics in automata theory as contained in [9,18]. In this paper we include some basic definitions and results for cover automata [3,4,5] to keep the paper self-contained.

We denote the number of elements of a finite set T by $\#T$ and an alphabet, i.e., a finite non-empty set, by Σ. A word $w = w_1 \dots w_n$, $w_i \in \Sigma$, $1 \leq i \leq n$, is an element of the free monoid Σ^* and its length is $|w| = n$; the empty string is the word with no letters and is denoted by ε, $|\varepsilon| = 0$.

The set of all words of length k over the alphabet Σ is denoted by Σ^k. The length of the longest word that belongs to a language $L \subseteq \Sigma^\star$ is called the *rank* of L.

A *deterministic finite automaton* (shortly, a DFA) A is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where:

- Q is the finite set of states;
- Σ is the input alphabet;
- $\delta : Q \times \Sigma \longrightarrow Q$ is the state transition function;
- $q_0 \in Q$ is the starting state, and
- $F \subseteq Q$ is the set of final states.

The *size* of a DFA is the number of states of the DFA.

A *cover automaton* for a language L with words of length less than or equal to l is a DFA A accepting a cover language L', i.e., a language with the property that $L' \cap \Sigma^{\leq l} = L$. Thus, a word w is in L if and only if it is accepted by A (as a DFA), and its length is less than or equal to l; in other words, a DFCA will accept a language, which may be potentially infinite, that "covers" the initial language. It is obvious that any DFA for a finite language L is also a DFCA, thus the number of states of a minimal DFCA for a finite language L is always smaller than the size of the minimal DFA for L. The state complexity of a language is the size of the minimal DFA accepting the language.

Several minimization algorithms for DFCAs have been developed in the past ten years [3,2,15,10], the best one having an $O(n \log n)$ time complexity.

It is important to note that there is more than one minimal deterministic finite cover automaton for a given finite language L, but all of them have the same number of states.

3 Enumeration of Finite Languages

We start our study by enumerating finite languages.

Let $string(n)$ be the n-th string over Σ in quasi-lexicographical order, e.g., in case of a binary alphabet $\Sigma = \{a, b\}$, that is

$string(0) = \varepsilon$,
$string(1) = a$, $string(2) = b$,
$string(3) = aa$, $string(4) = ab$, $string(5) = ba$, $string(6) = bb$,
$string(7) = aaa$, $string(8) = aab$, $string(9) = aba$,
$string(10) = abb$,$string(11) = baa$, $string(12) = bab$, $string(13) = bba$, $string(14) = bbb$,
$string(15) = aaaa$,. . . ,
$string(2^n - 1) = \underbrace{a \dots a}_{n}$, $string(2^n) = \underbrace{a \dots a}_{n-1}b$, . . . , $string(2^{n+1} - 2) = \underbrace{b \dots b}_{n}$.

In general, $string(n) = string(c_0 2^0 + c_1 2^1 + \dots c_m 2^m)$, where $\sum_{i=0}^{m} c_i 2^i = n$, thus $m = \min\{u \mid 2^{u+1} > n\}$.

We establish the following one to one correspondence between binary strings with no leading zeros and finite languages:

if $L = \{string(i_1), \dots, string(i_n)\}$, where $0 \leq i_1 < i_n$, then the binary string defined by its digits

$$digit(x) = \begin{cases} 1 & \text{if } x = i_j \text{ for some } 1 \leq j \leq n \\ 0 & \text{otherwise,} \end{cases}$$

can be used to encode L. Thus, we establish a complete enumeration of finite languages using binary representation of natural numbers.

For example, the language $\{\varepsilon, a, abb\}$ over $\Sigma = \{a, b\}$ can be encoded by:

abb	aba	aab	aaa	bb	ba	ab	aa	b	a	ε
1	0	0	0	0	0	0	0	0	1	1

i.e., by 10000000011.

In other words, $string(n) \in L$, iff the n-th bit of the number representing L is 1.

Definition 1. *[12] Let $D = \{w_0, w_1, w_2, \ldots\}$ be an infinite language , e.g., to be used for describing objects, the words are listed in order. The* redundancy *of D is the function*

$$\rho_D(n) = |w_n| - (\lfloor \log_k n \rfloor + 1),$$

where k is the alphabet size.

The quantity $\rho_D(n)$ is the number of extra digits required to denote the n-th object using the string $w_n \in D$, instead of using the standard digital representation of the number n with no leading zeros.

Our encoding for finite languages is optimal according to this definition, since it is a bijection between finite languages and binary strings (natural numbers written in base 2).

For a prefix-free language L, if a word w is in the language, then any prefix of w, as well as any word of the form wu, is not in L. Translating this in terms of the enumerating function $string$, we observe that if $w = string(n) \in L$, then $string(2n+1) \notin L$ and $string(2n+2) \notin L$, since $string(2n+1) = string(n)a$ and $string(2n+2) = string(n)b$. Of course, we have to exclude the prefixes of the current word $string(n)$, i.e., $v = string((\lfloor (n+1)/2 \rfloor) - 1) = string(\lfloor (n-1)/2 \rfloor) \notin L$ ($w = va$ or $w = vb$).

Example 1. If $bb = string(6) \in L$, then $string(2 \cdot 6 + 1), string(2 \cdot 6 + 2) \notin L$, i.e., $string(13) = bba$, and $string(14) = bbb$ are not in the language L.

Indeed, we can reiterate this condition and get that $string(2 \cdot 13 + 1), string(2 \cdot 13 + 2), string(2 \cdot 14 + 1), string(2 \cdot 14 + 2) \notin L$, i.e., $string(27), string(28), string(29), string(30) \notin L$, i.e., $string(2(2 \cdot 6 + 1) + 1), string(2(2 \cdot 6 + 1) + 2), string(2(2 \cdot 6 + 2) + 1), string(2(2 \cdot 6 + 2) + 2) \notin L$.

For prefix-closed languages is exactly the opposite case, if $string(2n+1) \in L$, then $string(n) \in L$; also, if $string(2n+2) \in L$, then $string(n) \in L$. Thus, if $string(n) \in L$, then $string((\lfloor (n+1)/2 \rfloor) - 1) = string(\lfloor (n-1)/2 \rfloor) \in L$.

For suffix-free languages, if $string(n) \in L$, and $|string(n)| = m$, then $string(n + k2^m) \notin L$.

Also, if $string(n) \in L$, and $|string(n)| = m$, $string(n - 2^m) \notin L$. Thus, $string(n - 2^m - 2^{m-1} - \ldots - 2^{m-k}) \notin L$, for all $1 \leq k \leq m$.

Balanced languages are those for which $||w|_a - |w|_b| \leq 1$.

Therefore, for a string of length n, we must have $\lceil \frac{n}{2} \rceil$ 1's and $\lfloor \frac{n}{2} \rfloor$ 0's or $\lceil \frac{n}{2} \rceil$ 0's and $\lfloor \frac{n}{2} \rfloor$ 1's, that is

$2 \cdot \frac{n!}{\frac{n}{2}!(l - \frac{n}{2})!}$ if n is odd and $\cdot \frac{n!}{(\frac{n}{2}!)^2}$ if n is even.

For balanced languages, we may use a different enumeration based on the Parikh function: first, we order strings over Σ by length, then by the number of occurences of b, then lexicographically.

Consequently, we obtain the following enumeration:

$\varepsilon, a, b,\ \ aa, ab, ba, bb, aaa, aab, aba, baa,\ \ abb, bab, bba, bbb,\ \ aaaa, aaab, aaba, abaa,\ \ baaa, aabb, abab, abba, baab, baba,\ bbaa, abbb, babb,\ bbab, bbba, bbbb.$

Recall that a finite language is represented by a natural number, where its binary representation is the characteristic vector for the above enumeration.

We can see that the following numbers represent balanced words: $0, 1, 2, 4, 5, 7 \ldots, 14, 22 \ldots, 27$. Thus, for any words of length l, among all 2^l, they are $2 \cdot \frac{l!}{\frac{l}{2}!(l - \frac{l}{2})!}$ if l is odd, and $\cdot \frac{l!}{(\frac{l}{2}!)^2}$ if l is even, balanced words, and they are exactly the ones in the middle of the interval.

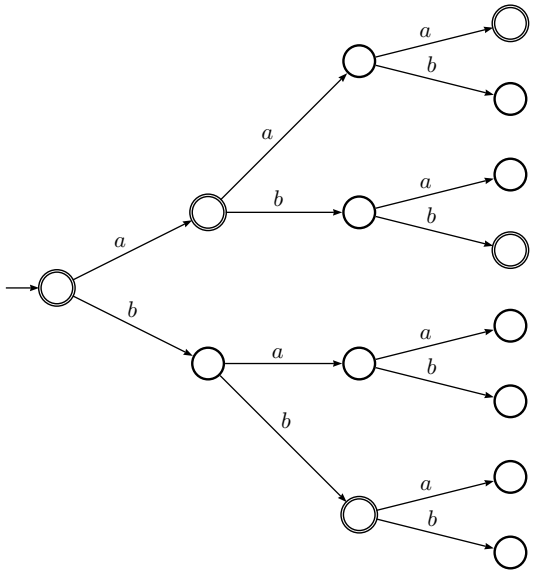
Our infinite language used to encode all finite languages is exactly the set of digital representations of all nonnegative integers, so the redundancy of that encoding is zero and, therefore, this encoding is also optimal.

4 Trie Uniform Random Generation

As it is evident from the previous section, the number of finite languages with a given rank l grows exponentially. Indeed, the number of non-empty finite languages over an alphabet Σ and of a given rank $l > 0$ is $\prod_{i=0}^{l-1} 2^{\#\Sigma^i}(2^{\#\Sigma^l} - 1)$. For $\#\Sigma = 2$, we have $2^{2^l-1}(2^{2^l} - 1)$, and the first values are 6, 120, 32640, 2147450880, 9223372034707292160, ... (it is sequence $A040996$ in Sloane' OEIS [17]). Thus, even considering small ranks, the exact enumeration of all languages is impractical. Statistically accurated results can be obtained if uniformly random samples are considered. As the probability of any individual member of the universe being selected is exactly the same as any other individual member, a uniform random generator produces a true, unbiased, random sample.

Considering the language enumerations presented in Section 3 suitable integer uniform random generators can be defined to generate finite binary languages. However, because we want to have the languages readly represented by a DFA we will consider yet another representation for finite languages.

Every non-empty finite language L can be represented by a trie [13]. A *trie* is a *tree*-shaped complete DFA where the initial state is the root of the tree and from each state the transitions from each different letter goes to different states. Thus, the number of final states corresponds exactly to the number of words in L. For example the language $L = \{\varepsilon, a, bb, aab, abb\}$ can be represented by the following trie:



There is a straightforward bijection between the tries over a binary alphabet and the finite languages as enumerated in Section 3. For instance, the trie above corresponds to the string 1101000011.

We now describe trie uniform random generators for finite languages and for each of the subfamilies considered in Section 3.

Each random generator generates languages of a given rank $l > 0$ ensuring (or not) that at least one word of length l occurs in the language. Although in Section 3 only binary alphabets were considered the uniform random generators will apply to languages over an alphabet of any size $\#\Sigma > 0$. We note that as we are not constrained to generate objects of a fixed size, all these generators are relatively simple. Also, we do not need to consider general techniques based on the *recursive method* for decomposable structures or on the enumeration of context-free languages [7,14,11].

A *general uniform random generator* for finite languages of rank $l > 0$ can easily be implemented using a depth-first construction of its trie. Each state has a probability of $\frac{1}{2}$ of being final, and at least a leaf has to be final.

A random generator for a subfamily of finite languages has to reflect the subfamily properties. The following subfamilies were considered: prefix-free, prefix-closed, suffix-free, suffix-closed, balanced and unbalanced.

A *uniform random generator for prefix-free languages* has to ensure that if a state is final all its descendants are non final (so the trie can be pruned at that state). A *suffix-free random generator* can be obtained considering the reversal languages.

A *uniform random generator for prefix-closed languages* has to ensure that if a state is final all its ascendants are final. Again, a *suffix-closed random generator* can be obtained considering the reversal languages.

For *balanced languages*, the random generator has an extra data structure that keeps the number of occurrences of letters in each path along the trie. This information is used for allowing or not a state to be final. The same data structure is used for unbalanced languages. We define a *unbalanced language* as one where the ratio of the number of occurrences of one letter to the number of occurrences of all the others is greater or equal than a certain value $r > 1$. In this case, the ratio r must also be considered by the generator.

All these generators were implemented within the FAdo system [6] and are available in version 0.9.3 or later.

5 Experimental Results

The main goal of this paper is the study of the compression ratio of DFCAs to minimal DFAs based on statistical methods. We consider sets of finite languages by rank. For each class of finite languages (of a given rank) we uniform random generated samples of $10,000$ tries. The size of each sample is sufficient to ensure the statistical significance with a 95% confidence level within a 1% error margin. It is calculated with the formula $n = (\frac{z}{2\epsilon})^2$, where z is obtained from the normal distribution table such that $P(-z < Z < z)) = \gamma$, ϵ is the error margin, and γ is the desired confidence level.

We used the FAdo and the Grail+ [8] symbolic computation systems. The FAdo system provides an interface to Grail+ that allows the easily call of Grail+ commands and the conversion between several finite automata representations. For each generated finite language the minimal DFA and a minimal DFCA were computed, and their size ratio compared. The minimal DFCA was computed using the Grail+ command `fmtofcm2` that implements the algorithm described in [2]. All other algorithms used were implemented in FAdo.

Figures 1–4 summarize the results obtained. The experiments were performed for languages with rank (i.e. the length of the longest word) l between 3 and 13. Figure 1 presents the compression ratio of DFCAs for general finite languages, as well as the maximum and minimum compression observed. These results are quite surprising. The compression ratio, on average, seems to diminish very quickly for relatively small ranks ($l \leq 13$). However, for ranks between 7 and 11 higher compressions are obtained. To see if some values were very far from the average we calculated the standard deviation of the samples. As Figure 2 shows that is not the case: the standard deviation is in general very small.

Figure 3 presents the compression ratio for free and closed languages. The results for both prefix and suffix languages were the same so we only show the values for prefix ones. The compression ratio for free languages is slightly better than for closed languages. Again the low compression ratio was not excepted. Note that it is easy to find "fix"-free(-closed) for which the compression by DFCAs is very high (for instance, the family $\sum_{1<i\leq m} a^i b$, for $m > 1$).

Finally, in Figure 4 we present the results for balanced and unbalanced languages. In general the behaviour of the compression ratio for these languages is near the one for general languages. For balanced languages with odd ranks the compression ratio outperforms the general case. For unbalanced as the ratio gets high the behaviour of the compression rate approaches the general case.

6 High Compression Languages

Let L be an infinite language recognized by a DFA A with n states and $l \in \mathbb{N}$ be such that $l > c \cdot n$, and for the language $L_l = L \cap \Sigma^{\leq l}$ we have that $L_l \cap \Sigma^l \neq \emptyset$, for some $c \in \mathbb{N}$, $c > 1$. Then $L_l = L \cap \Sigma^{\leq l}$ is a finite language for which L is a cover language. The language L_l contains a word of length l, thus the minimal DFA recognizing L needs at least $l + 1$ states. Since A is a cover automaton for L_l

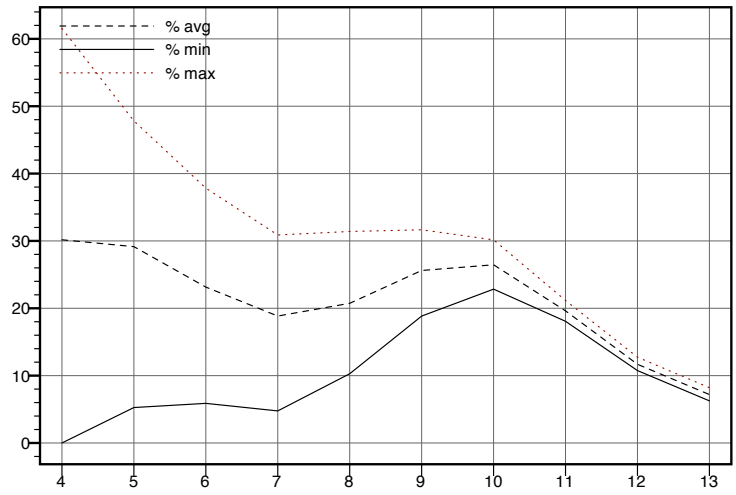


Fig. 1: Maximum, average and minimum gain of DFCAs for general finite languages

it follows that a minimal deterministic cover automaton for L_l has at most n states, therefore the compression ratio is at least c.

This very simple result proves that there is a very big family of languages for which we can obtain arbitrary large compression ratios, in spite of the fact that in general we do not obtain big compression ratios.

We conducted the following experiment:

1. We uniformly random choose cyclic DFA's (recognizing infinite languages) with n states.
2. Then compute the minimal DFA and minimal DFCA for L_l and determine the compression ratio.

For the random generation of DFAs we used the generator implemented in FAdo and described in [1]. As before we considered samples of $10,000$ DFAs. The tests considered DFAs over a binary alphabet with $n = 10$ and $n = 20$, and $1 \leq c \leq 5$. The average size of minimal DFCAs was 9.8 and 19.82, respectively. The size of the minimal DFAs depend on the value of l. For $n = 20$, for instance their average values are between 288.7 and 1845.5, for $20 \leq l \leq 100$. Figure 5 shows the compression ratios, that indeed are very high.

7 Conclusions

We did extensive experiments using combined software FAdo and Grail+ to obtain a complete statistical analysis of the expected compression ratio between the size of the minimal DFA and the size of a corresponding minimal DFCA.

The results show us with a error margin of 1%, that for binary languages, the expected reduction in the number of states is negligible. However, we give a very big class of languages, where the compression ratio can be arbitrary large. This yields to the conclusion that using a statistical analysis approach is not enough to say that DFCAs are in general not smaller than DFAs with respect to state complexity, for most cases, but we should be aware that there exist important classes of languages where the compression can be significant and we should investigate these classes in more detail.

It would be interesting to study the same problem from the topological point of view, for example, to check the density of the languages having no compression. Another approach is to study the problem using the measure theory, to determine the size of the set of incompressible languages, or of the languages where the compression ratio is in a given interval.

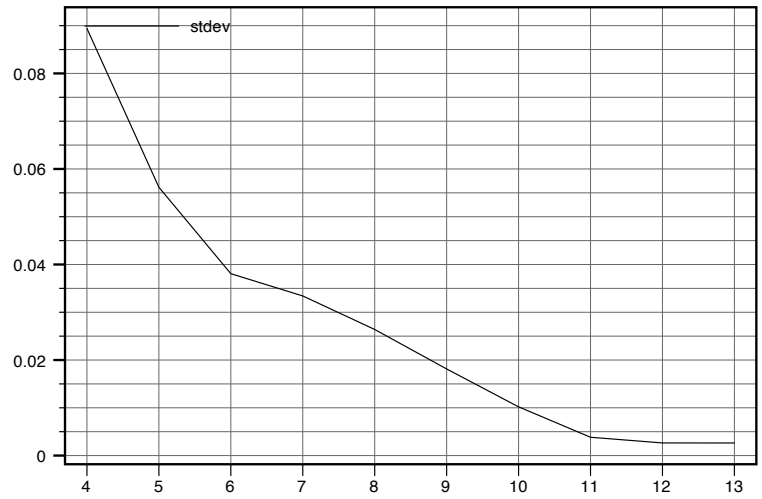


Fig. 2: Standard deviation of the compression ration for general finite languages

8 Acknowledgements

Nelma Moreira and Rogério Reis thank the *Fundação para a Ciência e Tecnologia* (FCT) and Program POSI for the funding throught project CANTE (PTDC/EIA-CCO/101904/2008).

References

1. Almeida, M., Moreira, N., and Reis, R. Enumeration and generation with a string automata representation. *Theoret. Comput. Sci. 387*, 2 (2007), 93–102.
2. Câmpeanu, C., Păun, A., and Yu, S. An efficient algorithm for constructing minimal cover automata for finite languages. *Int. J. Found. Comput. Sci. 13*, 1 (2002), 83–97.
3. Câmpeanu, C., Santêan, N., and Yu, S. Minimal cover-automata for finite languages. In *Workshop on Implementing Automata* (1998), J.-M. Champarnaud, D. Maurel, and D. Ziadi, Eds., vol. 1660 of *Lecture Notes in Computer Science*, Springer, pp. 43–56.
4. Câmpeanu, C., Sântean, N., and Yu, S. Minimal cover-automata for finite languages. *Theor. Comput. Sci. 267* (September 2001), 3–16.
5. Câmpeanu, C., and Păun, A. Counting the number of minimal dfca obtained by merging states. *Int. J. Found. Comput. Sci.* (2003), 995–1006.
6. FAdo: tools for formal languages manipulation. `http://fado.dcc.fc.up.pt/`, Access date:1.03.2011.
7. Flajolet, P., Zimmermann, P., and Cutsem, B. V. A calculus for the random generation of labelled combinatorial structures. *Theor. Comput. Sci. 132*, 2 (1994), 1–35.
8. The Grail+ Project. A symbolic computation environment for finite state machines, regular expressions, and finite languages. Available online at the address: `http://www.csd.uwo.ca/Research/grail`.
9. Hopcroft, J. E., and Ullman, J. D. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
10. Körner, H. A time and space efficient algorithm for minimizing cover automata for finite languages. *International Journal of Foundations of Computer Science (IJFCS) 14*, 6 (2003), 1071–1087.
11. Lee, J., and Shallit, J. Enumerating regular expressions and their languages. In *Proc. of CIAA 2004, Kingston, Canada* (2004), M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, Eds., vol. 3317 of *LNCS*, Springer, pp. 2–22.
12. Levenshtein, V. I. On the redundancy and delay of decodable coding of natural numbers. *Problemy Kibernet. 20* (1968), 173–179.

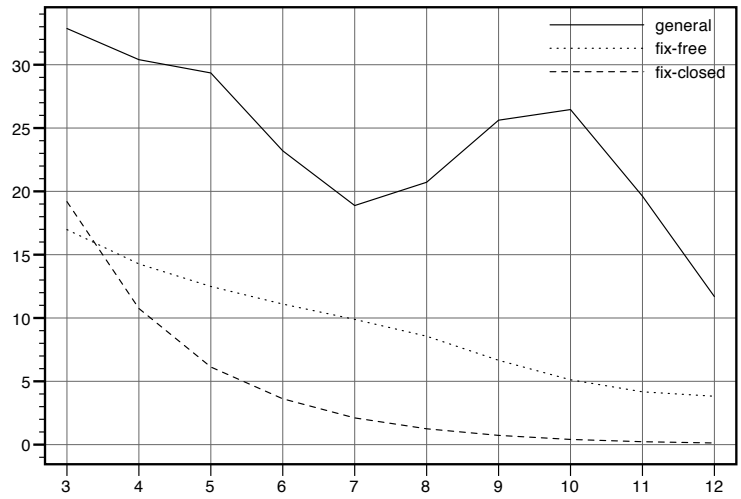


Fig. 3: Compression ratio for (suf/pre)fix-free and (suf/pre)fix-closed finite languages

13. Lothaire, M. Algorithms on words. In *Applied Combinatorics on Words*, no. 105 in Encyclopedia of Mathematics and its Applications. CUP, 2005, ch. 1.
14. Mairson, H. G. Generating words in a context-free language uniformly at random. *Information Processing Letters 49* (1994), 95–99.
15. Păun, A., Sântean, N., and Yu, S. An $O(n^2)$ algorithm for constructing minimal cover automata for finite languages. In *CIAA'00* (2000), pp. 243–251.
16. Sântean, N. Towards a minimal representation for finite languages: Theory and practice. Master's thesis, Department of Computer Science, The University of Western Ontario, 2000.
17. The On-line Encyclopedia of Integer Sequences. `http://oeis.org/`, Access date:1.03.2011.
18. Yu, S. Regular languages. In *Handbook of formal languages, vol. 1.* Springer-Verlag, New York, NY, USA, 1997, pp. 41–110.

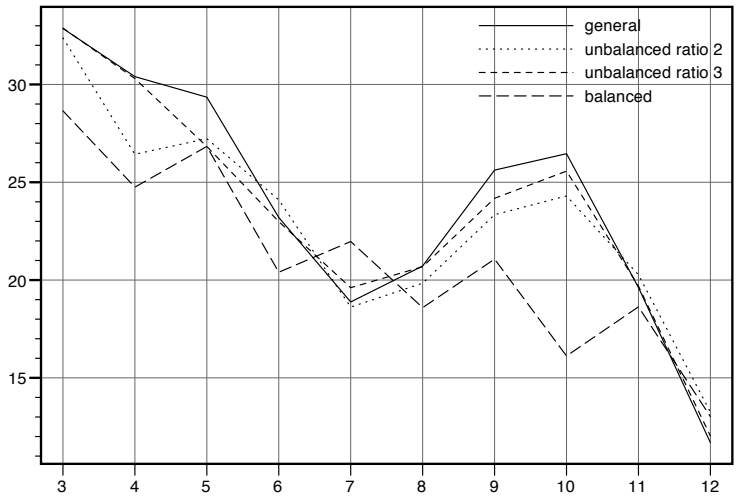


Fig. 4: Compression ratio for balanced and unbalanced finite languages

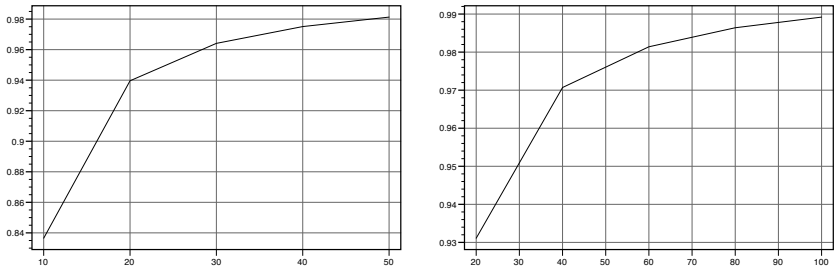


Fig. 5: Compression ratio for $L(A)_l$ with $n = 10$ (left) and $n = 20$ (right), and $l = nc$, for $1 \leq c \leq 5$.