Genetic Algorithms in Syllable-Based Text Compression*

Tomáš Kuthan and Jan Lánský

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
tkuthan@gmail.com, zizelevak@gmail.com

Abstract. Syllable based text compression is a new approach to compression by symbols. In this concept syllables are used as the compression symbols instead of the more common characters or words. This new technique has proven itself worthy especially on short to middle-length text files. The effectiveness of the compression is greatly affected by the quality of dictionaries of syllables characteristic for the certain language. These dictionaries are usually created with a straight-forward analysis of text corpora. In this paper we would like to introduce an other way of obtaining these dictionaries – using genetic algorithm. We believe, that dictionaries built this way, may help us lower the compress ratio. We will measure this effect on a set of Czech and English texts.

1 Introduction

In the early times of the computer age memory and storage capacity were highly limited and extremely expensive. This brought great pressure on storing the data as dense as possible and therefore created ideal conditions for data compression. But due to the fascinating development of computers, we have been witnessing for several last decades, storage capacity grew rapidly while it's price went down in a similar manner. Common hard disk drive of today's PC could easily carry all the code of all the computers in the early 70's. It could seem, that in this situation there is no need for effective compression. But together with the sources the demand raised as well. The amount of data companies deal with and want to or need to archive is unimaginable. Every percentage spared has it's immediate value in money. Another good example is networking. The dynamics of the growth of the network capacity does not even resemble the numbers we are used to by storage. It is very reasonable to transport data compressed to save the bandwidth.

Now that we explained the importance of data compression, we will try to specify the structure of the the coded files. Generally we recognize two types of files - binary and text files. There are many algorithms for binary compression including the whole are of lossy compression, but this concern is beyond the area

* This research was partially supported by the Program "Information Society" under project 1ET100300419.

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 21–34, ISBN 80-7378-002-X.

of interest of this paper. The structure of a text file depends on it's language. We may assume, that two documents in the same language have similar structure. Two different languages may have several similar characteristics. We may ask, whether both languages have rich morphology, or whether they for example both have fixed word-order. In languages with rich morphology the words usually consist of several syllables. Here syllables play the role of natural transition between letters and words.

The size of the file is an other aspect. Experience shows us, that character-based compression is more successful with smaller files, while word-based compression works better with larger files. Syllable-based methods have shown good results when used on middle-size to small documents.

This can be quite important with the already mentioned networking. File sizes of most common network content - *html pages* - are rather smaller. This creates ideal circumstances for syllable based compression methods.

Another important aspect of syllable based compression are the dictionaries of frequent syllables. These are used to initialize the compress algorithms data structures and greatly influence their effectiveness, especially in the early phase of the compression. Later the effect of the already processed part of input dominates over the effect of the initial settings. Therefore the role of dictionaries is vital with smaller files and slightly looses it's importance on bigger files. But, as mentioned above, it is the area of small to middle-size files, where the syllable-based methods are targeted.

Building an optimal dictionary is not an easy task. Including too many rare syllables into the dictionary results in longer bit-codes of the more common ones and hence longer coded message. On the other hand not including a frequent syllable means, that it would have to be coded by symbols (which is expensive) and initialised with low frequency and accordingly longer bit-code. The number of unique syllables in one language is measured in tens of thousands and every single one of them can be either included or excluded. That brings us in front of a problem of finding optimal solution among 2^N candidates, where N is the number of unique syllables. Genetic algorithm is a search technique, which can be employed for exploring such big, highly non-linear spaces.

2 Syllable based approach to text compression

In his study from 2005 [9], Jan Lánský has introduced a new approach to compression by symbols, the syllable-based compression. This new concept led to designing two new algorithms, which had good results in practical use and under certain circumstances even outperformed such sophisticated programs as bzip2. This chapter is dedicated to presenting this technique.

2.1 Languages and syllables

Knowing and understanding the structure of coded message can be very helpful in designing new compression method. In our case the coded message is a text

in natural language. It's structure is determined by the characteristic of the particular language. One linguistic aspect is the morphology. Languages with richer morphology (Czech, German, Turkish) tend to creating new words and word-forms by concatenating the root of the word with one or several prefixes or suffixes. On the other hand in languages like English the same effect is achieved by accumulating words. In the first category of languages we may find (thanks to their agglutinative nature) many rather long words composed of higher number of syllables. Such words are not very common in English. We can expect, that syllable-based compression will give better results on the first group of languages.

What is actually a syllable? Usually it is presented as a phonetic phenomenon. American Heritage Dictionary [11] gives us the following definition: 'A unit of spoken language consisting of a single uninterrupted sound formed by a vowel, diphthong, or syllabic consonant alone, or by any of these sounds preceded, followed, or surrounded by one or more consonants.' Correct *Hyphenation* (decomposition of a word into syllables) is a highly non-trivial issue. It can depend on he etymology of the certain word and there can be several different ways, how to do it. Fortunately we do not have to decompose the word always correctly according to the linguistic rules. We have to decompose it into substrings, which appear relatively often in the language. At this purpose we can get by with the following definition: 'Syllable is a sequence of sounds containing exactly one maximal subsequence of vowels'[1].

We recognize five basic categories of syllables: *capital* (consist of upper-case letters), *small* (lower-case letters), *mixed* (first letter is upper-case, other lower-case), *numeric* (numeral characters) and *other* (characters other than letters and numbers). Capital, small and mixed syllables altogether are called *literal* syllables, while numeral and other are called *non-literal*.

2.2 Hyphenation algorithms

To perform syllable-based compression, we need a procedure for decomposition into syllables. We will call an algorithm *hyphenation algorithm* if, whenever given a word of a language, it returns it's decomposition into syllables. According to our definition of syllable every two different hyphenation of the same word always contain the same number of syllables. There can be an algorithm, that works as a hyphenation algorithm for every language. Then it is called *universal hyphenation algorithm*. Otherwise we call it specific *hyphenation algorithm*.

We will describe four universal hyphenation algorithms: universal left P_{UL}, universal right P_{UR}, universal middle-left P_{UML} and universal middle-rightP_{UMR}.

The first phase of all these algorithms is the same. Firstly, we decompose the given text into words and for each word mark it's consonants and vowels. Then we determine all the maximal subsequences of vowel. These blocks form the ground of the syllables. All the consonants before the first block belong to the first syllable and those behind the last block will belong to the last syllable.

[1] for formal definitions concerning languages and syllables see [9]

Our algorithms differ in the way they redistribute the inner groups of consonants between the two adjusting vowel blocks. P_{UL} puts all the consonants to the preceding block and P_{UR} puts them all to the subsequent block. P_{UML} and P_{UMR} try to redistribute the consonant block equally. If their number is odd P_{UML} pushes the bigger partity to the left, while P_{UMR} to the right. The only exception is, when P_{UML} deals with an one-element group of consonants. It puts the only consonant to the right to avoid creation of not so common syllables beginning with a vowel.

Example 1. Hyphenating *priesthood*

correct hyphenation	priest-hood
universal left P_{UL}	priesth-ood
universal right P_{UR}	prie-sthood
universal middle-left P_{UML}	priest-hood
universal middle-right P_{UMR}	pries-thood

We have measured the effectiveness of these algorithms. In general, P_{UL} was the worst one; it had lowest number of correct hyphenations and produced largest sets of unique syllables. The main reason for this was, that it generates a lot of vowel-started syllables, which are not very common. P_{UR} was better but the most successful were both 'middle' versions. English documents were best hyphenated by P_{UMR}, while with Czech texts P_{UML} was slightly better.

In the following few paragraphs we will describe two syllable-based compression methods

2.3 LZWL

LZWL is a syllable version of well-known LZW algorithm [14]. The algorithm uses a dictionary of phrases, which is implemented by a data structure called *trie*. Each phrase is assigned an ordinal number according to time, when it was inserted into the dictionary.

During initialization this structure is filled with small syllables from the dictionary of frequent syllables. In each step we identify the maximal syllable chain, that forms a phrase from dictionary and at the same time matches a prefix of the not yet processed part of input. The number of the phrase (or better to say it's binary representation) is printed on the output. It could happen, that this maximal chain equals empty syllable. This would mean, that there is a new syllable on the input and we would have to encode it character by character.

Before performing the next step we add a new phrase into the dictionary. This new phrase is constructed by concatenating the phrase from last step with the first syllable of the current phrase.

For more information on LZWL algorithm please consult [9].

2.4 HuffSyllable

HuffSyllable is a statistical syllable-based text compression method. This technique was inspired by the principles of HuffWord algorithm [15].

It uses *adaptive Huffman tree* [7] as it's primary data structure. For every syllable type there is one tree built. In the initialization phase the tree for small syllables is filled with frequent syllables from the database together with their frequencies. In each step of the algorithm we try to estimate the type of next syllable. If the type is different than anticipated, binary code of an *escape sequence* assigned to the correct syllable type is printed on the output. Next, the code of the syllable is printed and it's frequency value in the tree is increased by one.

When the number of incrementations reaches certain value, the actualization of frequencies takes place. All the values are halved, which enforces rebuilding the whole tree.

For more information on HuffSyllable see [9].

2.5 Dictionaries

As we see, both algorithms use dictionaries of frequent syllables during the initialization. As we already mentioned in the first section, these dictionaries have crucial effect on the effectiveness of the algorithm, especially when compressing small files. These dictionaries are obtained by analysing a specimen of texts in the given language. There are two ways of doing it described in [8] - *cumulative* and *appearance* approach.

The cumulative criterion says, that a syllable is characteristic for the language, if it's quotient of occurrence to the number of all syllables in the texts is higher than certain rate. Acronym *C65* stands for dictionary containing all the syllables having the quotient higher than 1/65000. On the other hand, building an appearance dictionary means including all the syllables, for which the number of documents, where they occurred at least once, is higher than certain percentage. Experimental results proved, that the use of appearance dictionaries gave slightly better results.

Both methods have their advantages and their draw-backs. In fourth section we will describe technique based on evolutionary algorithms, which take both aspects into account.

3 Genetic algorithms in text compression

In 1997, Üçolük and and Toroslu have published an article about use of genetic algorithm in text compression. Ideas presented in our paper are strongly influenced by the results of their research, so let us give a brief summary of their method.

Üçolük and Toroslu have studied compression based on Huffman encoding upon mixed alphabet of characters and syllables[2]. This alphabet is apparently a subset of union of all letters and syllables. The issue is, which syllables should

[2] note, that this is a slightly different approach, than the one we are using. In their concept, rare syllables are dissolved into characters every time, they occur in the coded message, raising the occurrence of it's characters. In contrast, when we come

be included to ensure the optimal length of the compressed text. Observations suggested, that including nearly all the syllables usually led to best results. To prove this theory, the whole power set of the set of all syllables had to be examined. A genetic algorithm has been designed for this task.

Nice overview of genetic algorithms can be found in [5]. The general principles are well known: Candidate solutions are encoded into individuals called *chromosomes*. Chromosomes consist of *genes*, each encoding particular attribute of the candidate solution. The values each gen can have are called *alleles*. The encoding can be done in several different ways: *binary encoding*, *permutational encoding*, *encoding by tree*, and several others. A population of individuals is initiated and then bred to provide an optimal solution. The breeding is performed by two genetic operators – *cross-over*, in which the two selected chromosomes exchange genes, and *mutation*, where the value of a random gene is switched. The quality of a candidate solution is represented by so-called *fitness*. Fitness has influence on the probability, that the chromosome will be selected for mating. The higher the value of the fitness function, the better the solution and the better chance, that genes of the individual will carry over into next generations. After certain amount of generations the algorithm should converge to the optimum.

In this particular case the candidate solution is represented by a binary string, where the value 1 of i-th position means including the i-th syllable in the alphabet and 0 excluding it. The fitness represents the length of the text, if it was coded by Huffman encoding above the candidate alphabet. But performing compression and measuring the compressed text length would be rather expensive; it would require the Huffman tree construction which is known to be of order $O(N \log N)$ with considerably large multiplicative constant. Therefore it was decided rather to estimate this value theoretically. This can be done in linear time.

The approximation is grounded on two facts. The first fact can be deduced from Shannon's contribution [6]:

Lemma 1. *If the entropy of a given text is H, then the greatest lower bound of the compression coefficient μ for all possible codes is $H/\log m$ where m is the number of different symbols of the text.*

Second, the Huffman encoding is optimal. This means the ratio of Huffman compression can be well estimated as

$$\mu = -\frac{1}{\log m} \sum_{i=1}^{m} p_i \log p_i \tag{1}$$

where p_i is the probability of the i-th symbol of alphabet to occur in the text. Having the compression ratio makes it easy to compute the final code length simply by multiplying it by the bit-length of the uncompressed text, which is

across a new syllable, we encode it character by character and add it into the set of syllables. Next time we read this syllable on input, we treat it just like any other syllable.

$n \log m$. After a little mathematical brushing up we get this formula as the desired approximation:

$$l = n \log n - \sum_{i=1}^{m} n_i \log n_i \tag{2}$$

4 Characteristic syllables and their determination by GA

We have already mentioned, how important the dictionaries of characteristic syllables were for the compression ratio. We have also made clear, that the construction of these dictionaries is a difficult issue. In this section we will finally introduce a genetic algorithm designed for this task.

The input of this algorithm is a collection of documents in given language, so-called *training set*. The algorithm returns a file containing the characteristic syllables as it's output. The encoding of candidate solutions into chromosomes is again very straightforward; provided that the training set contains a set of N unique syllables, every individual is represented by a binary string of length N, where the value 1 on i-th position means including i-th syllable in the set of characteristic syllables, while 0 means excluding it. The role of the fitness function is played by estimated compressed length of a specimen from the training set. We are breeding the population to find a solution minimizing this value.

Algorithm 1 shows, how the evaluation of characteristic syllables works.

Algorithm 1 Genetic algorithm for characteristic syllables

```
syllable space initialization
generate random initial population
while not last generation do
  select several texts for specimen
  new generation ← empty set
  while size of new generation ≤ POOLSIZE do
    A ← random individual from old generation
    B ← another random individual from old generation
    C ← cross-over(A,B)
    add C into new generation
  end while
  if best individuals of old generation are better than worst new individuals then
    replace up to KEEPRATE worst new individuals with best old individuals
    /*application of elitism*/
  end if
  switch generations
  mutate random individual
end while
```

Our fitness function tries to approximate resulting bit length of the text compressed by HuffSyll algorithm. The behaviour of this algorithm enables us

to compute this value theoretically and therefore in reasonable time. The resulting dictionary of characteristic syllables should be optimal for use with HuffSyll. It will be interesting to examine, whether this dictionary introduces some improvements of the LZWL effectiveness too.

4.1 Evaluating fitness

The most important part of a genetic algorithm is the fitness function. It has to be accurate enough to provide good ordering on the set of candidate solutions and it has to be efficient, because it is called very often. The requirements concerning speed do not allow us using sophisticated calculations with high complexity.

We have decided not to use the whole training set in the fitness evaluation, but rather it's subset. For each generation we randomly select a specimen and use it for computing the fitness of all individuals. This attitude has two advantages: first, the evaluation needs less time, and second, the appearance of the syllable in the language is taken in concern. It does not only matter, how many occurrence the syllable has in the training set, but also in how many texts it appears at least once, and therefore how big the chance is, that it will appear in the specimen. After experimenting with the specimen size, we agreed on specimen consisting of five documents.

The most accurate way of evaluating fitness would be performing the actual compression and measuring the resulting file size. Again, this would be unacceptably time-consuming. We had to do an approximation similar to the one mentioned in last section.

The contribution of the characteristic syllables to the estimated bit length may be evaluated by a formula very similar to formula 2. The only difference is, that we will not only work with syllable frequencies in the file, which compressed bit length we are trying to estimate, but also with their frequencies in the whole training set. We will refer to these global numbers as n'_i for number of occurrences of i-th syllable and n' for the number of all syllables in the training set. Our new formula will be as follows

$$l = n \log n' - \sum_{i=1}^{m} n_i \log n'_i \tag{3}$$

The situation will be slightly different with the syllables marked as rare (non-characteristic). These syllables would have to be encoded character by character in the compression. They would be initialized with lower frequency, too. We take this into account in our approximation by adding an estimate of bits necessary for encoding the syllable and by increasing it's code bit length by one.

The principals of the fitness evolution are outlined in pseudo code in algorithm 2.

4.2 Setting parameters

The behaviour and effectiveness of a genetic algorithm depends on the settings of several parameters. These parameters include size of the population, probability

Algorithm 2 Evaluation of fitness

```
R ← 0
for all file in specimen do
  N' ← 0, S ← 0, P ← 0
  for all syllable in set of syllables do
    V ← number of occurrences of syllable in file
    V' ← number of occurrences of syllable in all the files
    N' ← N' + V'
    if syllable is marked as characteristic then
      S ← S + V * lg2(V')
    else if V > 0 then
      S ← S + V * (lg2(V') − 1)
      P ← P+ estimated bit length of syllable's code
    end if
  end for
  N ← number of syllables in file
  R ← R + N * lg2(N') − S + P
end for
return R
```

of cross-over, probability of mutation, number of generations, range of elitism and degree of siding with better individuals in selection. There is no general rule for setting these parameters. The situation is even more complicated by the fact, that these parameters often act in a rather antagonistic manner.

Most authors writing about evolutionary computing agree, that among these parameters the one most important is the size of the population. Population too small does not allow the algorithm to sufficiently seek through the whole search space. Inadequately large population leads to consuming too much computational power without much significant improvement in the quality of the solution. Optimal size depends on the *nature* of the problem and on it's *size*[3]. Yong Gao insists, that the dependency with size is linear [4]. We have experienced good results with populations of several hundreds individuals.

One thing that is tight very closely to population size is the type of cross-over. In [10] the advantages of different types of cross-overs (one-point, two-point, multi-point and uniform) are discussed. We have decided for multi-point cross-over, because of it's positive effect, when used with smaller populations. It prevents the algorithm from creating unproductive clones. We have set the number of cross-over points to the value of 10.

Elitism is an instrument against loosing the best solution found so far. It means, that instead of replacing whole old population with the new one, we keep several members of the old population as long as they are better than the worst members of the new population. Too much elitism may cause *premature convergence*, which is a really unpleasant consequence. To avoid this, we restrict elitism to small number of individuals, about one percent of the population.

[3] size of problem is defined as length of candidate solution encoding

In selection, better individuals are treated with favor; better chromosome has higher chance to be chosen, than the one below standard. The probability p, that an individual is chosen, may be formalized by

$$p(x_i) = \frac{k - f(x_i)}{nk - \sum_{j=0}^{n-1} f(x_j)} \tag{4}$$

were n stands for population size, f for fitness and constant k is set equal to $max_{x \in P}(f(x)) + min_{x \in P}(f(x))$. P stands for the population.

5 Experimental Results

In this section we will present results of HuffSyll and LZWL algorithms when used with genetically determined dictionaries of common syllables. The algorithms will be compared according to resulting bpc[4] value. The test will be performed on two collections of files, one for English and one for Czech.

5.1 Training sets

We have constructed two training sets, which served us as input for the genetic algorithm. They were also used for obtaining cumulative dictionary *C65*, to which we compared the results of genetically determined dictionaries.

English set consised of 1000 documents randomly chosen from two corpora; 100 files from [2] and 900 law documents from [1].

Czech sets contained 69 middle-size (mean 215,3kB) fiction texts from [3]. The rest was formed by 931 newspaper articles obtained from Prague Dependency Treebank [12]. With mean of 1,8kB they were consider as short documents.

5.2 Test sets

Both set for testing had the size of 7000 documents. Czech set contained 69 texts from [3] and 6931 articles from [12]. English test set consisted of 300 short stories from [2] and 7000 documents randomly chosen from [1].

5.3 Results

In table 1 we can see the measured results for HuffSyll and LZWL algorithms with different hyphenation and with or without use of genetically determined characteristic syllables for Czech language. In table 2 there are the same data for English.

As we can see, with HuffSyll there is significant gain when using characteristic syllables instead of cumulative dictionary C65. This positive effect is greater for smaller files. It is not surprising, because as we have already mentioned, when compressing longer files the effect of already processed part of input overweights

[4] bits per character

Table 1. Effect of characteristic syllables in compression of Czech texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UL} + C65	5.32	4.70	4.18	3.95	3.89
HuffSyll + P_{UL} + GA	4.77	4.40	4.09	3.92	3.87
HuffSyll + P_{UML} + C65	5.32	4.67	4.10	3.85	3.80
HuffSyll + P_{UML} + GA	4.70	4.31	3.99	3.81	3.78
HuffSyll + P_{UMR} + C65	5.25	4.62	4.08	3.85	3.81
HuffSyll + P_{UMR} + GA	4.72	4.33	3.99	3.82	3.79
HuffSyll + P_{UR} + C65	5.29	4.64	4.09	3.84	3.80
HuffSyll + P_{UR} + GA	4.76	4.35	4.00	3.82	3.78
LZWL + P_{UL} + C65	6.16	5.19	4.29	3.80	3.54
LZWL + P_{UL} + GA	6.08	5.19	4.31	3.81	3.54
LZWL + P_{UML} + C65	6.30	5.19	4.24	3.75	3.52
LZWL + P_{UML} + GA	6.15	5.23	4.29	3.76	3.51
LZWL + P_{UMR} + C65	6.26	5.16	4.23	3.75	3.51
LZWL + P_{UMR} + GA	5.98	5.16	4.27	3.76	3.51
LZWL + P_{UR} + C65	6.30	5.19	4.24	3.75	3.52
LZWL + P_{UR} + GA	6.20	5.26	4.31	3.77	3.52

Table 2. Effect of characteristic syllables in compression of English texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UL} + C65	4.51	3.62	3.26	3.16	3.16
HuffSyll + P_{UL} + GA	3.90	3.36	3.18	3.14	3.15
HuffSyll + P_{UML} + C65	4.84	3.84	3.39	3.24	3.21
HuffSyll + P_{UML} + GA	4.04	3.46	3.26	3.20	3.20
HuffSyll + P_{UMR} + C65	4.79	3.82	3.39	3.25	3.18
HuffSyll + P_{UMR} + GA	3.98	3.45	3.25	3.21	3.17
HuffSyll + P_{UR} + C65	4.90	3.88	3.41	3.27	3.25
HuffSyll + P_{UR} + GA	4.00	3.46	3.25	3.22	3.23
LZWL + P_{UL} + C65	5.61	3.63	2.81	2.70	2.86
LZWL + P_{UL} + GA	5.43	3.69	2.86	2.73	2.87
LZWL + P_{UML} + C65	5.89	3.77	2.88	2.73	2.87
LZWL + P_{UML} + GA	5.30	3.63	2.87	2.74	2.87
LZWL + P_{UMR} + C65	5.87	3.79	2.89	2.74	2.89
LZWL + P_{UMR} + GA	5.25	3.57	2.84	2.74	2.88
LZWL + P_{UR} + C65	5.92	3.80	2.91	2.77	2.91
LZWL + P_{UR} + GA	5.25	3.59	2.87	2.76	2.91

the initial settings from dictionary. An important matter is, that although the gain gets smaller with lengthy texts, it never turns into draw-back. And the average improvement of 0.6bpc in the category of shortest Czech files and 0.8bpc in the same category of English files is noteworthy.

On the other hand, use of GA dictionaries did not bring such great effort by LZWL algorithm. There is some upturn in the categories of smaller files, but it is not as remarkable as we witnessed by HuffSyll. More important is, that in some file size categories GA dictionary gave actually slightly worse results, than ordinary C65 dictionary. To conclude, we may remark, that dictionaries genetically determined for use with HuffSyll did not astonished us when used with LZWL. There is no point in preferring them to cumulative dictionaries, which are much easier to obtain. To the contrary, if we already posses the GA dictionary for HuffSyll, we may use it for LZWL as well, without worrying that the results will be too bad.

We have also compared the effectiveness of the syllable-based methods to several commonly used compress programs, namely bzip2 1.0.3, gzip 1.3.5 and compress 4.2. For comparison we have chosen the most successful hyphenation algorithm and the most successful dictionary of characteristic syllables for the given language. Table 4 contains the results for Czech and table 3 for English. Figure 1 shows the comparison for czech documents graphically.

These numbers are slightly distorted by omission of the formats overheads. gzip uses 18B of additional data for header plus 32-bit CRC checksum, and bzip2 uses 12B of additional information. Regrettably we could not find the format specification for compress, but we assume, that the overhead is similar. There is no such additional information in files compressed by LZWL and HuffSyllable. Especially in the category of the smallest files this gives them an advantage over the others, but the effect is not so high.

Table 3. Comparison with commonly used compress programs - English texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UL} + GA	3.90	3.36	3.18	3.14	3.15
LZWL + P_{UL} + GA	5.43	3.69	2.86	2.73	2.87
gzip 1.3.5	4.94	3.05	2.38	2.58	3.10
bzip2 1.0.3	5.28	3.03	2.18	2.13	2.38
compress 4.2	5.86	4.19	3.44	3.25	3.31

As we see, the results of syllable-based text compression methods were not bad. In the category of smallest files, HuffSyll has fully taken the advantage of dictionary initialization and outmatched even such sophisticated program as bzip2. As long as larger texts are concerned, bzip2 complied to it's reputation, and prevailed in a convincing manner. Relatively worst results were reached by compress; it was outperformed by both syllable-based algorithms as well as by both competition programs used in production. gzip was outmatched by HuffSyll

Table 4. Comparison with commonly used compress programs - Czech texts

Method	100B-1kB	1-10kB	10-50kB	50-200kB	200kB-2MB
HuffSyll + P_{UML} + GA	4.70	4.31	3.99	3.81	3.78
LZWL + P_{UML} + GA	6.15	5.23	4.29	3.76	3.51
gzip 1.3.5	5.68	4.56	3.87	3.61	3.53
bzip2 1.0.3	6.05	4.62	3.54	3.06	2.83
compress 4.2	6.33	5.38	4.51	4.04	3.82

in the category of short documents, but was better with longer files. With LZWL it was exactly vice-versa; for larger files LZWL gave better results than gzip.

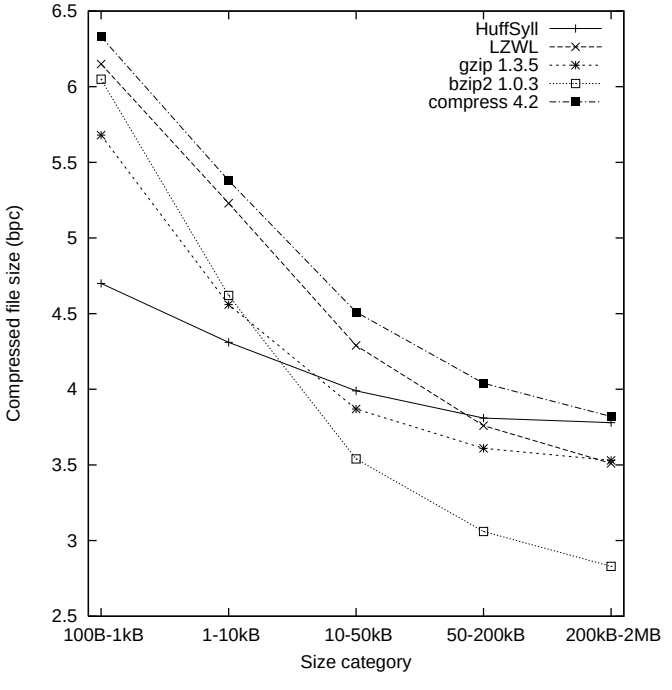

Fig. 1. Comparison with commonly used compress programs - Czech texts

6 Conclusion

We have introduced a new method for obtaining dictionaries of characteristic syllables for syllable-based text compression. On English and Czech texts, we

have documented it's advantages in comparison with cumulative dictionaries for HuffSyll. We have studied the gain it produces with respect to size of compressed files and hyphenation algorithm used. We have examined, how the use of HuffSyll optimized dictionary affects effectiveness of LZWL algorithm.

In future works, it could be interesting to design a genetic algorithm for obtaining dictionaries optimized for LZWL.

References

1. *California law* http://www.leginfo.ca.gov/calaw.html
2. *Canterbury corpus* http://corpus.canterbury.ac.nz
3. *e-knihy* http://go.to/eknihy as visited on 2nd February 2005
4. Gao, Y. Population Size and Sampling Complexity in Genetic Algorithms, *Proceedings of the Bird of a Feather Workshops (GECCO) – Learning, Adaptation and Approximation in Evolutionary Computation, 2003*
5. Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub. Co. 1989, ISBN 0201157675.
6. Hamming, R. W. *Coding and Information Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1986.
7. Huffman, D. A. *A method for the construction of minimum redundancy codes.* Proc. Inst. Radio Eng. 40:1098-1101, 1952
8. Lánský J., Žemlička M. *Compression of Small Text Files Using Syllables.* Technical report no. 2006/1. KSI MFF UK, Praha, January 2006.
9. Lánský J., Žemlička M. Text Compression Syllables. *Richta K., Snášel V., Pokorný J.: Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects.* CEUR-WS, Vol. 129, pg. 32-45, ISBN 80-01-03204-3.
10. Spears W. M., De Jong K. A. *An Analysis of Multi-Point Crossover.* FGA, (1991) 301–315
11. *The American Heritage® Dictionary of the English Language, Fourth Edition.* Houghton Mifflin Company, 2004. http://dictionary.reference.com/browse/syllable (accessed: January 09, 2007).
12. *The Prague Dependency Treebank* http://ufal.mff.cuni.cz/pdt/
13. Üçolük G., Toroslu H.: *A Genetic Algorithm Approach for Verification of the Syllable Based Text Compression Technique.* Journal of Information Science, Vol. 23, No. 5, (1997) 365–372
14. Welsh T. A. *A technique for high performance data compression.* IEEE Computer, 17,6,8-19,1984
15. Witten I., Moffat A., Bell,T.: *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, 1994