Text Compression: Syllables

Jan Lánský and Michal Žemlička

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
zizelevak@gmail.com, michal.zemlicka@mff.cuni.cz

Abstract. There are two basic types of text compression by symbols – in the first case symbols are represented by characters, in the second case by whole words. The first case is useful for very short files, the second case for very long files or large collections. We supposed that there exist yet another way where symbols are represented by units shorter than words – syllables. This paper is focused to specification of syllables, methods for decomposition of words into syllables, and using syllable-based compression in combination of principles of LZW and Huffman coding. Above mentioned syllable-based methods are compared with their counterpart variants for characters and whole words.

1 Introduction

Knowledge on structure of coded messages can be very useful for design of a successfull compression method. When compressing text documents, the structure of messages is dependent on used language. We can expect that documents written in the same language will have similar structure.

Similarity of languages can be seen according many aspects. Language classification can be made, for example, according their use of fixed or free word order or whether they have simple or rich morphology.

To the languages with rich morphology they belong for example Czech and German. In these languages is syllable a natural element logically somewhere between characters and words. Each word is often created by two or more syllables.

At the beginning we supposed that:

- Syllable-based compression will be suitable for middle-sized files, character-based compression will be more suitable for small files, and word-based compression will suit best for very large files.
- Syllable-based compression will be more suitable for languages with rich morphology.
- The number of unique syllables in given language is much lower than the number of unique words. This lead to lower memory requirements of syllable-based compression algorithms than of word-based ones.
- The sets of syllables of two documents written in the same language are usually more similar than sets of words of these documents.

K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 32–45, ISBN 80-01-03204-3.

2 Languages and Syllables

We can classify languages according their morphology. There are languages like English having simple morphology where from one stem there can be derived only a few word forms. There are also languages with rich morphology (like Czech) where from one stem it can be derived several hundred word forms.

We will demonstrate it on some examples from Czech and English. The verb *take* has in English only next 5 forms: *take*, *takes*, *taking*, *took*, *taken*. Czech verb *vzít*, which corresponds to the English verb *take*, has next 24 forms: *vzít*, *vzíti*, *vezmu*, *vezmeš*, *vezme*, *vezmeme*, *vezmete*, *vezmou*, *vzal*, *vzala*, *vzalo*, *vzali*, *vzaly*, *vezmi*, *vezměme*, *vezměte*, *vzat*, *vzata*, *vzato*, *vzati*, *vzaty*, *vzav*, *vzavši*, *vzavše*. Next difference is in creation of words with similar meaning and negations. In English there are used combinations of more words for getting different meaning, for example *get on*, *get off*. The negation in English is created by combination with word not, for example *not get on*. In Czech prefixes and suffixes are used instead. To English *get on*, *get off*, *not get on* correspond Czech *nastoupit*, *vystoupit*, *nenastoupit*. In Czech we can create from verb *skočit* (*jump* in English) using prefixes 9 next similar verbs: *přeskočit*, *nadskočit*, *podskočit*, *odskočit*, *rozskočit*, *naskočit*, *vskočit*, *uskočit*, *vyskočit*. For each of these verbs we can create their antonyms by using prefix *ne*: *neskočit*, *nepřeskočit*, *nenadskočit*, *nepodskočit*, *neodskočit*, *nerozskočit*, *nenaskočit*, *nevskočit*, *neuskočit*, *nevyskočit*. For each of these 20 verbs there exist 24 grammatical forms. So from this one word *skočit* we can derive over 400 similar words, but these words are composed from only a few tens of syllables.

2.1 Syllables

According Compact Oxford English Dictionary [10] syllable is defined as: 'A unit of pronunciation having one vowel sound, with or without surrounding consonants, and forming all or part of a word.'

As the decomposition to syllables is used in data compression, it is not necessary to decompose words into syllables always correctly. It is sufficient if the decomposition produces groups of letters that occur quite frequently. We therefore use simplified definition below that is not equivalent with the grammatically correct definition. 'Syllable is a sequence of sounds, which contains exactly one maximal subsequence of vowels.' This definition implies that the number of syllables in a word is equal to the number of maximal sequences of vowels in the same word. For example, the word *famous* contains two maximal sequences of vowels: *a* and *ou*, so this word is created from two syllables: *fa* and *mous*. Word *pour* contains only one maximal sequence of vowels *ou*, so whole this word is created by only one syllable.

Decomposition words into syllables is used for example for text formatting when we want to split word exceeding end of line. Disadvantage of this way is that we cannot decompose all words and some words must be left unsplitted.

One of the reasons why we selected syllables is that documents contain less unique syllables than unique words. Example Czech document (Karel Čapek:

Hordubal) with the size of 195 kB has 33,135 words form which are 8,071 distinct and 61,259 syllables from which are 3,187 distinct. English translation of bible [9] with the size of 4MB has 767,857 words from which are 13,455 distinct and 1,073,882 syllables from which are 5,604 distinct.

2.2 Problems in Decomposition of Words into Syllables.

Decomposition of words into syllables is not always unique. To determine it, we must often know origin of the word. Some problems will be demonstrated on selected Czech words. We supposed that for compression it is sufficient to use some approximation of correct decomposition of words into syllables. We supposed that this approximation would have only a small negative effect on reached degree of compression.

An example of non-uniqueness of decomposition of words into syllables is the word *Ostrava* which can be correctly decomposed into *Os-tra-va* and also into *Ost-ra-va*. Generally sequence of letters *st* is often a source of ambiguity of decomposition of Czech words to syllables.

An example of a variant decomposition of similar sequences of letters, which is caused by origin of words, is words *obletí* and *obrečí*, that have first two letters same. Word *obletí* (will fly around) was created by adding prefix *ob* to the word *letí* (flies). Word *obrečí* (will cry over) was created by adding prefix *o* to the word *brečí* (cries). So the word *obletí* is decomposed into *ob-le-tí*, word *obrečí* is decomposed into *o-bre-čí*. A big group of problems is brought by words of foreign origin and their adapted forms.

Sometimes it can by difficult to recognize real number of syllables of given word. Although the word *neuron* is a prefix of the word *neuronit*, these words have different decomposition to syllables. Word *neuron* is decomposed to *neu-ron*, word *neuronit* is decomposed to *ne-u-ro-nit*. In the first case the sequence of letters *neu* is composed by one syllable, in the second case it is composed from two syllables.

Full correctness of decomposition of words into syllables can be reached only at the price of very high effort. For the use in compression it is not important whether the decomposition is absolutely correct, but whether the produced groups of letters are frequent enough.

Other goal is to formalize terms letter, vowel, consonant, word, syllable, and language. When we try to define expressions vowel and consonant, we must interesting about position letter in the word. For example in Czech letters *r* and *l* can be according their context both vowels and consonants. In English similar role is played by the letter *y*.

2.3 Definition of Syllable.

Definition 1. *Let Σ be finite nonempty* set of symbols *(alphabet). Symbol $\lambda \notin \Sigma$ is called* empty word. *Let $\Sigma_{\mathrm{P}} \subseteq \Sigma$ be* set of letters, *then $\Sigma_{\mathrm{N}} = \Sigma \backslash \Sigma_{\mathrm{P}}$ is called* set of nonletters. *Let $\Sigma_{\mathrm{C}} \subseteq \Sigma_{\mathrm{N}}$ be* set of digits, *then $\Sigma_{\mathrm{Z}} = \Sigma_{\mathrm{N}} \backslash \Sigma_{\mathrm{C}}$ is called* set of special characters.

Definition 2. *Let Σ be finite nonempty set of symbols. Let $\Sigma_{\mathrm{P}} \subseteq \Sigma$ be set of letters. Let $\Sigma_{\mathrm{M}} \subset \Sigma_{\mathrm{P}}$ be set of small letters, then $\Sigma_{\mathrm{V}} = \Sigma_{\mathrm{P}} \backslash \Sigma_{\mathrm{M}}$ is called set of capital letters. If there exists bijection $\psi : \Sigma_{\mathrm{M}} \to \Sigma_{\mathrm{V}}$, then Σ_{P} is called* correct set of letters.

Note 1. The set of letters for most of natural languages is correct, but exists exceptions like German. In German letter ß can be according context or small or capital letter. If we want to work with non-correct sets of letters, we must modify definition 2.

Definition 3. *Let Σ be finite nonempty set of symbols. Let $\Sigma_{\mathrm{P}} \subseteq \Sigma$ be set of letters.*

Let $\phi : (\Sigma \cup \{\lambda\}) \times \Sigma_{\mathrm{P}} \times (\Sigma \cup \{\lambda\}) \to \{0, 1, 2, 3\}$ be a function. Let $\beta \in \Sigma_{\mathrm{P}}$, let $\alpha, \gamma \in (\Sigma_{\mathrm{P}} \cup \{\lambda\})$. Then:

- *If $\phi(\alpha, \beta, \gamma) = 0$ then β in context α, γ is called* vowel. *($\beta \in \Sigma_A$).*
- *If $\phi(\alpha, \beta, \gamma) = 1$ then β in context α, γ is called* consonant. *($\beta \in \Sigma_B$)*
- *If $\phi(\alpha, \beta, \gamma) = 2$ then β in context α, γ is consisting from vowel β_1 followed by consonant β_2. As β_1 and β_2 are together one letter, they cannot be split into different syllables.*
- *If $\phi(\alpha, \beta, \gamma) = 3$ then β in context α, γ is consisting from consonant β_1 followed by vowel β_2. As β_1 and β_2 are together one letter, they cannot be split into different syllables.*

Note 2. It is probable that there exist languages where we do not need to know context α, γ to decide if letter β is a vowel or a consonant. In both Czech and English the use of context is necessary.

In Czech can letters r and l be according their context used as vowels or as consonants. If α or γ are vowels, then $\beta = r$ (respectively $\beta = l$) is a consonant, in the opposite case it is a vowel. Examples: *mlčet*, *mluvit*, *vrtat*, *vrátit*.

In English the letter y in context of two vowels has the role of a consonant (example: *buying*).

In English words of type trying $\phi(r, y, i)$ has value 2, so y is composed from vowel y_1 followed by consonant y_2.

We supposed that there exist a language where $\phi(\alpha, \beta, \gamma)$ can have value 3, but in English and Czech it is not so.

The size of context one from left and right side is sufficient for Czech and for most of English words. We suppose that this size of context can be sufficient too.

Definition 4. *Let Σ be a finite nonempty set of symbols. Let $\Sigma_P \subset \Sigma$ be a set of letters. Let $\Sigma_C \subset \Sigma_N$ be a set of digits. Let $\Sigma_C \cap \Sigma_P = \emptyset$. Let $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma$. If one of the following cases is valid, then α is called a* word over alphabet Σ.

If $\alpha_i \in \Sigma_Z$ for $i = 1, \ldots, n$, word α is called other.
If $\alpha_i \in \Sigma_C$ for $i = 1, \ldots, n$, word α is called numeric.
If $\alpha_i \in \Sigma_M$ for $i = 1, \ldots, n$, word α is called small.

If $\alpha_i \in \Sigma_V$ *for* $i = 1, \ldots, n$, *Word* α *is called* capital.
If $\alpha_1 \in \Sigma_V$ & $\alpha_i \in \Sigma_M$ *for* $i = 2, \ldots, n$, *word* α *is called* mixed.

Note 3. Numeric words and other-words are called together as words from non-letters. Small, capital, and mixed words are called as words from letters.

Definition 5. *Let* Σ *be finite nonempty set of symbols. Let* $\alpha = \alpha_1, \ldots, \alpha_n, \alpha_i \in \Sigma$. *Let* $\beta_1, ..., \beta_m$ *are words over alphabet* Σ. *Let* $\beta_i \cdot \beta_{i+1}$ *are not words over alphabet* Σ *for* $i = 1, \ldots, m-1$. *Let* $\alpha = \beta_1 \cdot ... \cdot \beta_m$. *Then* $\beta_1, ..., \beta_m$ *is called* decomposition of the string α into words.

Note 4. From definitions 4 and 5 follows that decomposition of strings into words is fast unique. There is an exception when after two ore more capital letters follows at least one small letter (example CDs), but this case is very rare in natural languages.

There exist two types of algorithms of decomposition of strings into words. Both types differ only in solving that exception. Algorithms of type A_1 create from this string capital-word and mixed-word (example C and Ds). Algorithms of type A_2 create from this string capital-word and small-word (example CD and s).

Words are decomposed into syllables (see Def. 9).

Definition 6. *Language* L *is an ordered 6-tuple* $(\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$, *where*

- Σ *is a finite nonempty set of symbols.*
- $\Sigma_P \subset \Sigma$ *is a correct set of letters.*
- $\Sigma_C \subset \Sigma_N$ *is a set of digits, where* $\Sigma_N = \Sigma \backslash \Sigma_P$.
- $\Sigma_M \subset \Sigma_P$ *is a set of small letters.*
- $\phi : (\Sigma \cup \{\lambda\}) \times \Sigma \times (\Sigma \cup \{\lambda\}) \to \{1, ..., 4\}$ *is a function which according definition 3 specify whether letter* β *in context* α, $\gamma \in (\Sigma \cup \{\lambda\})$ *is a vowel or a consonant.*
- A *is an algorithm which for each string* $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma$ *finds some decomposition of given string into words.*

Definition 7. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let* $\alpha, \gamma \in \Sigma_B^*$, $\beta \in \Sigma_A^*$, $|\beta| \in \{1, 2, 3\}$. *If* $\alpha \cdot \beta \cdot \gamma$ *is a word over alphabet* σ, *then it is syllable of the language* L.

Note 5. Notation $\alpha \in \Sigma_B^*$ (respective $\beta \in \Sigma_A^*$) mean that β is a sequence of consonants (respective vowels).

From definitions 4 and 7 follows that each syllable is also a word. So we will recognize five types of syllables: other syllables, numeric syllables, small syllables, capital syllables, and mixed syllables.

Condition that each syllable must be also word is necessary. For example, the string *xxAxx* is not word, therefore it cannot be (according Def. 7) syllable.

Definition 8. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let* $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma_P$ *is a word over alphabet* Σ.

If $(\exists k)\alpha_k \in \Sigma_A$*, then* α *is called* word decomposable into syllables.
If $(\forall k)\alpha_k \in \Sigma_A$ *then* α *is called* non-syllable word.

Definition 9. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let* $\alpha = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma_P$ *is a word decomposable into syllables. Let* $\beta_1, ..., \beta_m$ *be syllables of the language L. Let* $\alpha = \beta_1 \cdot ... \cdot \beta_m$*, then* $\beta_1 \cdot ... \cdot \beta_m$ *is called* decomposing word α into syllable.

Definition 10. *Let* $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ *be a language. Let P be an algorithm which input is document D decomposed into words by algorithm A into words* $\alpha_1, ..., \alpha_n$ *over alphabet* Σ*. If for all* α_i *they are valid both the following conditions, then P is called* algorithm of decomposition into syllables for language L.

- *If* α_i *is a word decomposable into syllables, then output of the algorithm is a decomposition of the word* α_i *into syllables.*
- *If* α_i *is a non-syllable word, numeric-word, or other-word, then output of the algorithm is a word* α_I*.*

Definition 11. *Let P be an algorithm of decomposition into syllables for language* L_1*. If B is an algorithm of decomposition into syllables for all other languages L, then we say that P is a universal algorithm of decomposition into syllables. If there exist a language* L_2 *for which P is not algorithm of decomposition into syllables, then we say that P is* specific algorithm of decomposition into syllables.

Note 6. Each universal algorithm of decomposition into syllables P has to use all information from definition of language L. In the other case we can construct language for which P will not be an algorithm of decomposition into syllables.

2.4 Examples of languages

There are two examples of languages: English and Czech. Czech languages has in comparison with English new diacritical letter.

English language can be characterized as $\mathrm{L_{EN}} = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ where:

- Σ = ANSI character set
- $\Sigma_P = \{a, \ldots, z, A, \ldots, Z\}$
- $\Sigma_C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\Sigma_M = \{a, \ldots, z\}$
- ϕ is defined as:
  - $(\forall \alpha \in (\Sigma \cup \{\lambda\}), \forall \beta \in (M \backslash \{y, Y\}), \forall \gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 0$
  - $(\forall \alpha \in (\Sigma \cup \{\lambda\}), \forall \beta \in (\Sigma_P \backslash M), \forall \gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 1$
  - $(\forall \alpha \in ((\Sigma \backslash M) \cup \{\lambda\}), \forall \beta \in \{y, Y\}, \forall \gamma \in ((\Sigma \backslash M) \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 0$
  - $(\forall \alpha \in (\Sigma_P \backslash M), \forall \beta \in \{y, Y\}, \forall \gamma \in M)\ \phi(\alpha, \beta, \gamma) = 2$
  - $(\forall \alpha \in M, \forall \beta \in \{y, Y\}, \forall \gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 1$
  - $(\forall \beta \in \{y, Y\}, \forall \gamma \in M) \phi(\alpha, \beta, \gamma) = 1$
  - where $M = \{a, e, i, o, u, y, A, E, I, O, U, Y\}$

– $A = A_1$ from Note 4.

Czech language can be characterized as $\mathrm{L_{CZ}} = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ where:

– Σ = ANSI character set
– Σ_P = {a,. . . ,z,A,. . . ,Z,á,č,ď,é,ě,í,ň,ó,ř,š,ť,ú,ů,ý,ž,Á,Č,Ď,É,Ě,Í,Ň,Ó,Ř,Š,Ť,Ú,Ů,Ý,Ž}
– Σ_C = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
– Σ_M = {a,. . . ,z,á,č,ď,é,ě,í,ň,ó,ř,š,ť,ú,ů,ý,ž}
– ϕ is defined as:
  - $(\forall\alpha \in (\Sigma \cup \{\lambda\}), \forall\beta \in M, \forall\gamma \in (\Sigma \cup \{\lambda\}))\ \phi(\alpha, \beta, \gamma) = 0$
  - $(\forall\alpha \in (\Sigma\ M), \forall\beta \in \{r, l, R, L\}, \forall\gamma \in ((\Sigma \cup \{\lambda\}) \backslash M))\ \phi(\alpha, \beta, \gamma) = 0$
  - else $\phi(\alpha, \beta, \gamma) = 1$
  - where M = {a,á,e,é,ě,i,í,o,ó,u,ú,ů,y,ý,A,Á,E,É,Ě,I,Í,O,Ó,U,Ú,Ů,Y,Ý}
– $A = A_1$ from Note 4.

2.5 Algorithms of Decomposition Into Syllables

We describe four universal algorithms of decomposition into syllables: universal left $\mathrm{P_{UL}}$, universal right $\mathrm{P_{UR}}$, universal middle-left $\mathrm{P_{UML}}$, universal middle-right $\mathrm{P_{UMR}}$. These four algorithms are called as algorithms of class $\mathrm{P_U}$. The names of these algorithms are derived from way these algorithms work. Inputs of these algorithms are language $L = (\Sigma, \Sigma_P, \Sigma_C, \Sigma_M, \phi, A)$ and document $D = \alpha_1, \ldots, \alpha_n$, $\alpha_i \in \Sigma$. These algorithms are composed from two parts. The first part is an initialization and this is common for all algorithms of the class $\mathrm{P_U}$. The second part is different for each algorithm.

At the beginning of the initialize part we decompose document D into words by algorithm A. Algorithm of class $\mathrm{P_U}$ is processing single words. Words from non-letters are automatic declared as syllables. For each word from letters is according function ϕ decided whose its letters are consonants and which are vowels. Maximal blocks (blocks that cannot be extended) of vowels are found afterwards. Blocks of vowels longer than three are not usually in natural languages, so maximal length of block of vowels is set to three. These blocks of vowels will create bases of syllables. For each block of vowels we must keep in memory its begin and end. Consonants, which are in the word before first block of vowels, are added to this block. Consonants, which are in the word after last block of vowels, are added to this block.

Single algorithms of class $\mathrm{P_U}$ are different in the way of adding consonants, which are between two blocks of vowels. After these ways of adding are algorithms named.

Algorithm universal left $\mathrm{P_{UL}}$ adds all consonants between blocks of vowels to the left block.

Algorithm universal right $\mathrm{P_{UR}}$ adds all consonants between blocks of vowels to the right block.

Algorithm universal right $\mathrm{P_{UMR}}$ in the case of $2n$ (even count) consonants between blocks adds to both blocks n consonants. In the case of $2n + 1$ (odd

count) consonants between blocks it adds to the left block n consonants and to the right block $n+1$ consonants.

Algorithm universal right P_{UML} in case of $2n$ (even count) consonants between blocks adds to both blocks n consonants. In the case of $2n+1$ (odd count) consonants between blocks it adds to the left block $n+1$ consonants and to the right block n consonants. The only exception from this rule is the case when between blocks it is only one consonant, this consonants is added to the left block.

Example: We will decompose word *priesthood* into syllables. We are using language L_{EN}. Blocks of vowels are (in order): *ie*, *oo*.

correct decomposition into syllables:	priest-hood
universal left P_{UL}:	priesth-ood
universal right P_{UR}:	prie-sthood
universal middle-left P_{UML}:	priest-hood (correct form)
universal middle-right P_{UMR}:	pries-thood

3 Compression methods

We used hypothesis that compressed text is structured into sentences and it is described by following rules. A sentence begins with mixed word and ends with other word, which contains a dot. Inside the sentences are switching regularly small words and other words. If the sentence begins with capital-word, then inside are switching regularly capital-words and other-word. Numeric-words appear rarely and are usually followed by other-words.

When we decompose words into syllables, we have problem with this model. Each word has different count of syllables. Small-word is usually followed by other-word, whereas small-syllable can be followed not only by other-syllables but also by another small-syllable.

To improve a compression of alphabet of syllables (or words) we have created for each language a database of frequent words. More details will be in section 4. Words from this database are used for initialization of compressing algorithms. When coding alphabet of given document we can code only words, which are not from out databases of frequent words. This is useful for smaller documents, on the bigger documents is that effect lower.

3.1 LZWL

Algorithm LZW [6] is a dictionary compression character-based method. Syllable-based version of this method has been named LZWL. Algorithm LZWL can work with syllables obtained by all algorithms of decomposition into syllables. This algorithm can be used for words (see Def. 4) too.

First we shortly recall classical method LZW [6]. Algorithm is using dictionary of phrases, which is represented by data structure trie. Phrases are numbered by integers afterwards order of adding.

In initialization step the dictionary is filled up with all characters from alphabet. In each next step it is searched for maximal string S, which is from dictionary and matches the prefix of still non-coded part of the input. Number of phrase S is sent to the output. A new phrase is added to the dictionary. This phrase is created by concatenation of string S and character that follows after S in file. Actual input position is moved forward by the length of S.

Decoding has only one situation for solving. We can receive number of phrase, which is not from dictionary. In this case we can create that phrase by concatenation of the last added phrase with its first character.

Syllable-base version is working over alphabet of syllables. In initialization step we add to the dictionary empty syllable and small syllables from database of frequent syllables. Finding string S and coding its number is analogical with character-based version, only that string S is a string of syllables. Number of phase S is encoded to output. It is possible that string S can be empty syllable. If S is empty syllable, then we must get from file one syllable called K and encode K by methods for coding new syllables, see section 4.2. Syllable K is added to dictionary. Actual position in the file is moved forward by the length of S, in the case when S is empty syllable, the input position is moved forward by the length of K.

In adding a phrase to dictionary there is a difference to character-based version. Phrase from the next step will be called S_1. If S and S_1 are both non-empty syllables, then we add new phrase to the dictionary. New phrase is created by concatenation S_1 with the first syllable of S. This solution has two advantages. The first advantage is that strings are not created from syllables that appear only once. Second advantage is that we cannot receive in decoder number of phrase that is not from dictionary.

3.2 HuffSyllable

HuffSyllalbe is statistical compression method based on adaptive Huffman coding and using structure of sentence in natural language. The idea of this algorithm was inspired by HuffWord [7]. Algorithm LZWL can work with syllables obtained by all algorithms of decomposition into syllables mentioned above. This algorithm can be used for words too.

For each type of syllables (small, capital, mixed, number, other) it is build adaptive Huffman tree [2], which is coding syllables of given type. In the initialization step of algorithm we add to Huffman tree for small syllables all syllables and their frequencies from database of frequent syllables.

In each step of the algorithm it is calculated expected type of actually processed syllable K. If syllable K has different type than it is expected, then an escape sequence is generated. Syllable K is encoded by Huffman tree corresponding to the syllable type. Calculating of expected type of syllable uses information from encoded part of input. We need to know the type of last syllable. If the last syllable is other-syllable, then it is known that this syllable contains a dot and that the type of the last syllable is letter-syllable.

Table 1. Expected types of syllables according type of previous syllable.

previous / expected type of syllable	Expected syllable
small	small
capital	capital
mixed	small
number	other
other syllable without dot, last syllable from letters is not capital	small
other syllable with dot, last syllable from letters is not capital	mixed
other, last syllable from letters is capital	capital

4 Technical Details

We supposed that all natural languages have their own characteristical set of syllables. Its approximation for English and Czech was created by set of testing documents. We created for each language and algorithm of decomposition into syllables one database of frequent syllables from letters. For each language we also created databases of frequent other syllables. Condition for adding syllable to database was that its frequency is greater than 1 : 65,000. Each database of syllables from letters contains approximately 3,000 syllables, whose sum of frequency is 96–98 % of all syllables. Each database of other syllables contains approximately 100 syllables, which sum of frequency is 99.5 % of all syllables.

These databases are used in initializations steps of compressing algorithms. This can improve compression ratio on smaller documents.

Although we have database of frequent syllables, sometimes we receive syllable K, which is not from this database and we must encode it. They are two basic ways. The first way is to encode K as code of length of K followed by the codes of individual characters from the syllable. The second (and better) way is to encode K as code of syllable type followed by code of length of K and codes of individual characters. We use the second way, because domain of coding function for distinct characters is given by the type of syllable and as it is smaller than in the first way. Numeric syllables are coded differently.

Encoding type of syllable depends on types of previous syllable and other criteria as in HuffSyllable. Length of codes for each types are 1, 2, 3, and 4. Average code length is 1.5 bits.

For encoding length of syllables are used two static Huffman trees, the first one for letter-syllables and the second one for other-syllables. Trees are initialized from statistics received from text documents.

For encoding distinct characters there are used two adaptive Huffman trees, the first one for syllables from letters and the second one for other syllables.

Numeric-syllables are coded differently from other types of syllables. We discover that numbers in text are naturally divided into a few categories. The first category contains small numbers (1–100), the second category represent year (1800–2000), in the third category there are very large numbers (for example

5,236,964) that usually have separated groups of digits to blocks by three. But these large numbers are decomposed into numeric-words and other-words. So we set maximal length of numeric-word to 4, longer numeric-words are split. For coding number of digits 2-bits binary coding is used. For coding distinct digits binary phased coding [4] is used.

5 Experimental Results

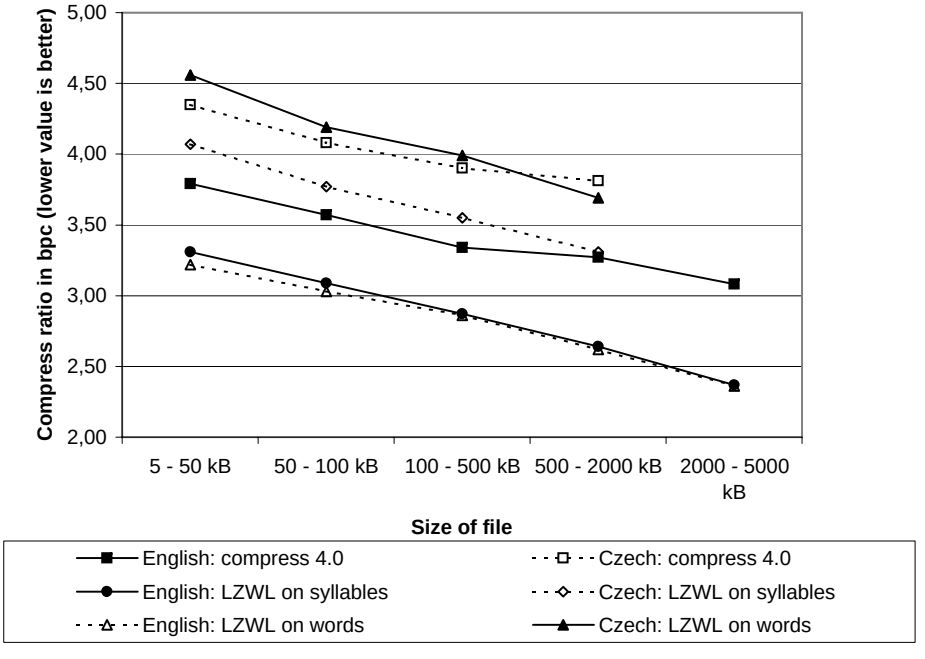

Fig. 1. Comparison of LZW-based methods on English and Czech

For testing there were used two sets of documents in plain text format. The first set contains 69 documents in Czech with total size of 15 MB. Most of these documents were received from [11]. The second set contains 334 documents in English with total size of 144 MB. In this set there are documents from project Gutenberg [12] and `bible.txt` from Canterbury corpus [9]. From each file form project Gutenberg there were removed first 12 kB of information about project because it was the same in all documents.

Table 2. Comparison of compression ratio in bits per character on English documents

Method\file	5–50 kB	50–100 kB	100–500 kB	500–2000 kB	2000–5000 kB
LZWL+P_{UL}	3.31	3.09	2.87	2.64	2.37
LZWL+P_{UR}	3.36	3.14	2.92	2.69	2.39
LZWL+P_{UML}	3.32	3.10	2.88	2.65	2.38
LZWL+P_{UMR}	3.32	3.10	2.89	2.66	2.38
LZWL(words)	3.22	3.03	2.86	2.62	2.36
compress 4.0	3.79	3.57	3.34	3.27	3.08
HS+P_{UL}	3.23	3.18	3.15	3.10	2.97
HS+P_{UR}	3.30	3.26	3.22	3.18	3.03
HS+P_{UML}	3.26	3.22	3.19	3.15	3.02
HS+P_{UMR}	3.27	3.23	3.20	3.16	3.02
HS(words)	2.65	2.58	2.52	2.38	2.31
ACM(words) [3]	2.93	2.74	2.55	2.35	2.27
FGK [2]	4.59	4.60	4.60	4.58	4.54
bzip2 [8]	2.86	2.60	2.40	2.21	2.03

Table 3. Comparison of compression ratio in bytes per character on Czech documents

Method\file	5–50 kB	50–100 kB	100–500 kB	500–2000 kB	2000–5000 kB
LZWL+P_{UL}	4.14	3.83	3.59	3.34	—
LZWL+P_{UR}	4.07	3.77	3.56	3.32	—
LZWL+P_{UML}	4.07	3.77	3.56	3.31	—
LZWL+P_{UMR}	4.07	3.77	3.55	3.31	—
LZWL(words)	4.56	4.19	3.99	3.69	—
compress 4.0	4.35	4.08	3.90	3.81	—
HS+P_{UL}	3.97	3.89	3.89	3.81	—
HS+P_{UR}	3.86	3.79	3.80	3.75	—
HS+P_{UML}	3.86	3.79	3.80	3.74	—
HS+P_{UMR}	3.87	3.79	3.80	3.75	—
HS(words)	3.71	3.51	3.43	3.21	—
ACM(words)	3.83	3.50	3.29	3.14	—
FGK	4.97	4.95	5.00	4.99	—
bzip2	3.42	3.10	2.88	2.67	—

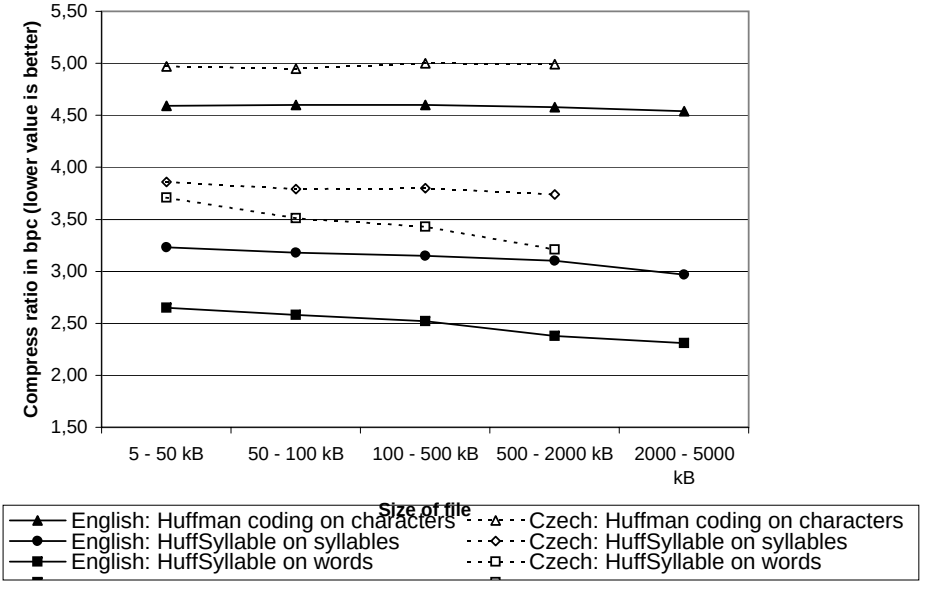

Fig. 2. Comparison of Huffman-based methods on English and Czech texts

6 Conclusion

In this paper we have introduced idea of syllable-based compression, its advantages and disadvantages. We have formally defined letters, syllable, words and algorithm of decomposition into words. We have introduced four universal algorithms of decomposition into words. We have created two syllable-based compression methods that use alternation of syllable types in sentences. The first method is based on algorithm LZW, the second on Huffman coding. The experimental results of these algorithms confirm our predictions, that tested syllable-based algorithms outperformed their character-based counterparts for both tested languages. Comparison of word-based and syllable-based versions of Huffman and LZW codings led to the result that in English the word-based versions of both algorithms outperform their syllable-based counterparts and in Czech the results are ambiguous: for Huffman coding word-beased version outperformed syllable-based one, for LZW coding the syllable-based one outperformed the word-based one.

In the future we want to decrease space and time requirements of implemented algorithms. We planned to addapt next syllable-based algorithms from their character-based versions, for example bzip2. We want to test our algorithms on more languages with rich morphology, for example on German and Hungarian.

References

1. D. A. Huffman. A method for the construction of minimum redundancy codes. *Proc. Inst. Radio Eng.*, 40:1098–1101, 1952.
2. D. E. Knuth. Dynamic Huffman coding. *J. of Algorithms*, 6:163–180, 1985.
3. A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16:256–294, 1998.
4. P. Elias. Universal codeword sets and representation of the integers. *IEEE Trans. on Information Theory*, 21(2):194–203, 1975.
5. S. W. Thomas, J. McKie, S. Davies, K. Turkowski, J. A. Woods, and J. W. Orost. Compress (version 4.0) program and documentation, 1985.
6. T. A. Welch. A technique for high performance data compression. *IEEE Computer*, 17(6):8–19, 1984.
7. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.
8. The bzip2 and libbzip2 official home page. http://sources.redhat.com/bzip2/ as visited on 6th February 2005.
9. Canterbury corpus. http://corpus.canterbury.ac.nz.
10. Compact Oxford English Dictionary. http://www.askoxford.com/ as visited on 3rd February 2005.
11. eknihy. http://go.to/eknihy as visited on 2nd February 2005.
12. Project Gutenberg. http://www.promo.net/pg.