Diacritics restoration based on word n-grams for Slovak texts

Article *in* Open Computer Science · January 2021

DOI: 10.1515/comp-2020-0143

5 authors, including:

Štefan Toth
University of Žilina
18 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Michal Ďuračík
University of Žilina
16 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Patrik Hrkút
University of Žilina
19 PUBLICATIONS **74** CITATIONS
SEE PROFILE

Matej Meško
University of Žilina
8 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Research Article

Štefan Toth*, Emanuel Zaymus, Michal Ďuračík, Patrik Hrkút, and Matej Meško

Diacritics restoration based on word n-grams for Slovak texts

https://doi.org/10.1515/comp-2020-0143
Received Apr 15, 2020; accepted Jun 27, 2020

Abstract: Despite the modern boom in technology, we are still faced with the fact that people write texts without diacritics. There are two main reasons for this. The first, historical reason stems from the past when the use of diacritics was troublesome and people would write text without them. The second one is the speed - typing without diacritics is usually faster. Text without diacritics is easy to understand for people, but for some types of documents, missing diacritics can cause a problem. This is also an issue when computers process such text. In this paper, we propose an algorithm based on word n-grams (a contiguous sequence of n words) that can restore diacritics of text written in the Slovak language. We also compare and evaluate our results with other algorithms developed for Slovak text.

Keywords: diacritic, diacritic restoration, n-grams, Slovak language, diacritical marks, accents

1 Introduction

The diacritic, defined by the Merriam-Webster dictionary [9], is a mark near or through an orthographic or phonetic character or combination of characters indicating a phonetic value different from that given the unmarked or otherwise marked element. Diacritical marks may appear above or below a letter, or in some other position such as within the letter or between two letters. The most common diacritical marks are called *accents*.

Diacritical marks are mainly required in official contacts, such as any e-mail communication between people who do not know each other, or official documents, such as a final thesis. However, the Slovak text is understandable without diacritical marks, and there are also other reasons why people in everyday life do not use the diacritics.

All of these reasons stem from the available technology – because hardly anyone omits diacritical marks in the text written on paper. The diacritics are often omitted in text found on mobile devices, such as mobile phones or tablets. When typing text on such devices, it takes more time to type a letter with an accent than without it, so the communication without the diacritics is faster.

Other reasons are historical - once accents were not well supported, for example, in the e-mail communication. Letters with accents were not encoded correctly. The recipient of such a message often received an incomprehensible set of squares instead of meaningful text and he had to guess what letter was intended by the sender. However, this problem is not an issue anymore; the internet communication is now able to handle the diacritics well. Another historical reason comes from the time when communication with a short text message (called SMS by Short Message Service) was used. The maximum length of one SMS is 140 bytes [6]. If the sender uses basic Latin characters without diacritics in a message, the default encoding is 7-bit (GSM-7). This means that the message can hold 160 characters. However, if a user uses non-standard or accented characters, the number of characters in a message will be truncated due to a change in the encoding (from 7-bit to 8bit or currently even 16bit for UCS2). For this reason, people preferred to write SMS messages without diacritics.

Another very current reason concerns mainly Slovaks working abroad. They often use a computer with a different language and another keyboard to communicate with their homeland, where "Slovak" letters may be missing. Of course, in terms of software, the Slovak keyboard can be installed, but these letters will not be indicated on the original keyboard.

To sum up, omitting diacritic was a practical and economical solution. But people have also realized that this makes typing messages faster.

Despite all these reasons, the diacritics in the Slovak language are important. One of the more poetic reasons is that we want to maintain a national diversity and uniqueness. But there is also a more pragmatic reason - processing text by computers. For example, missing diacritics could

***Corresponding Author: Štefan Toth:** Department of Software Technologies, Faculty of Management Science and Informatics, University of Žilina, Žilina, Slovakia; Email: stefan.toth@fri.uniza.sk
Emanuel Zaymus, Michal Ďuračık, Patrik Hrkút, Matej Meško: Department of Software Technologies, Faculty of Management Science and Informatics, University of Žilina, Žilina, Slovakia

make things more complicated if we design an artificial intelligence solution for text meaning recognition.

In this paper, we are focusing on the Slovak language and its specific diacritical marks. Table 1 shows all lowercase and uppercase letters with their diacritical marks in the Slovak alphabet.

Table 1: Diacritics in the Slovak language.

Diacritical mark	Slovak letters with diacritical marks
’ (acute accent)	á, é, í, Í, ó, ŕ, ú, ý Á, É, Í, Ĺ, Ó, Ŕ, Ú, Ý
ˇ (caron)	č, ď, dž, ň, ľ, š, ť, ž Č, Ď, DŽ, Ň, Ľ, Š, Ť, Ž
¨ (diaeresis)	ä, Ä
^ (circumflex accent)	ô, Ô

In addition to these letters, there may be other diacritical marks in the text that come from other languages. Indeed, texts can also contain various letters with diacritical marks in foreign words (for example: ö, ü, ů, Ö, Ü, Ů, ã, Ã, ê, Ê, ć, Ć, ś, Ś, ç, Ç, ș, Ș, ț, Ț, . . .) [15].

There are many different types of languages in the world in terms of diacritics. Some of them contain new letters (for example, Slovak) while in others, the diacritics is used only to distinguish words that were taken over from other languages. For example, English words taken from another language that frequently appear with the diacritics include café, résumé or resumé (this helps distinguish from the verb resume).

The main problem with the restoration of diacritics in most languages is that the diacritics not only change the way a word is pronounced but often the meaning of the word as well. If there were just one way to transform a word without diacritics into a correct word, the reconstruction could be done using a dictionary. An example of such a word in the Slovak language is “bábätko” (translation: “a baby”). Removing diacritics, we obtain the word “babatko” as a result. For this particular word there is no other word with a different meaning in the Slovak language, so that word can be restored by adding a combination of diacritical marks. So, in this case, it is easy to restore correct diacritics.

The more difficult problem is a word that can have a different meaning by adding a various combination of diacritics. An example of such a word is “boli” (meaning “we were” or “they were”) and itself is a valid word. One meaning we can reconstruct from this word is “bolí” (meaning “it hurts”) and another one is “bôli” (meaning “in a grief”). We need context to reconstruct the diacritics correctly.

2 State of the Art

A great deal of effort is devoted to restoring diacritics in every language that uses diacritics. The reasons why people write text without diacritics are in most cases the same. There are many languages where diacritical marks are used. Publications dealing with this problem can be found for languages such as Czech [10], Hungarian [11], Romanian [13], Arabic [16], Croatian [8] and many others. Slovak is one of the Slavic languages and therefore techniques of the diacritics renewal in these languages are important for us as well.

Several papers have been published also for the restoration of diacritics for the Slovak language. For example, in [3] authors used *Hidden Markov Model and Viterbi algorithm*. In [7] authors used *statistical language models* of Slovak language. Nowadays, *deep neural networks* have shown remarkable results as authors in [5] and [10] show state of the art solutions.

Now, we are mainly focusing on the Slovak language. There are three freely accessible services for the reconstruction of diacritics on the internet. These services will be described in the following sections.

2.1 Diakritik by SAS

The *Diakritik* tool for reconstruction of diacritics was created by Ľ. Štúr Institute of Linguistics of the Slovak Academy of Sciences (SAS). It is based on a large corpus of Slovak texts and provides several methods to reconstruct diacritics with different ratios of error, speed and purpose: first match, random match, most used and words n-grams (2, 3, 4, 5 or 6 gram) and more. It has been available for public use since 18th August 2014 [14]. As the authors mention, the error rate of the reconstructed text, i.e. the ratio of words with incorrectly determined diacritics, is about 0.2%. Roughly one in five hundred words is reconstructed incorrectly.

2.2 Diacritics restoration by BRM

The *diacritics restoration* (in Slovak “Dopĺňač diakritiky”) (BRM) by R. Hraška is a web application that is freely available to the public [4]. The restoration is determined by the most frequent word from diacritics form. However, the ap-

plication does not use any sophisticated methods or algorithms or complicated linguistics methods.

2.3 Statistical diacritic restoration by STUBA

The *statistical diacritic restoration* system was created by J. Geder as his bachelor thesis [2] at the Faculty of Informatics and Information Technologies at the Slovak University of Technology in Bratislava (STUBA). The application creates all possible combination of n-words with correct diacritics. In the next step, it calculates the probability of occurrence for all n-words combinations and the one with the highest probability is chosen. The algorithm can reconstruct only words without a typo or missing letters. The success rate of the algorithm is 98%.

3 Proposed algorithm

Our proposed algorithm is based on comparing the surroundings of the word to which we want to add the diacritics with the n-grams associated with that word. We had 4-grams available as a data source, so the following algorithm will be described for n-grams where n is equal to 4. Therefore, we consider three words before and three words after this word in the context if these words exist. The scheme of the proposed algorithm is shown in Figure 1.

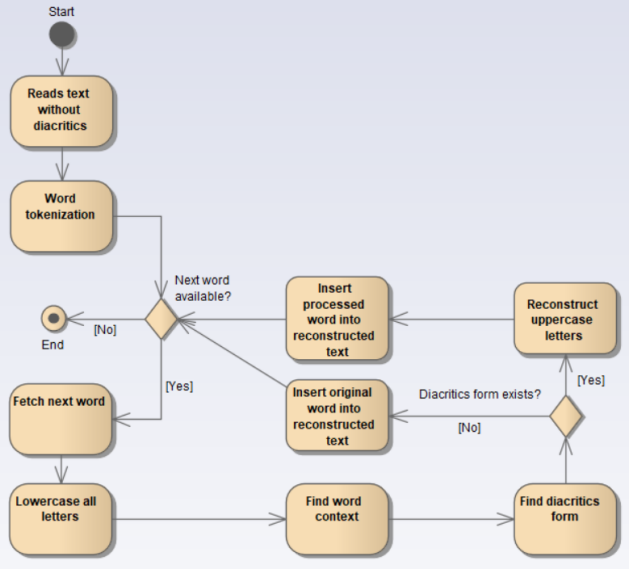

Figure 1: Activity diagram – base algorithm scheme

In the beginning, we read the input text for which we want to reconstruct the diacritics. Then we divide the input text into separate words (word tokenization). In order to preserve all the punctuation, white spaces, and other characters that the user types in our resulting reconstructed text, we must also save them. Then we process all the words one by one. If we still have a word, we take it and normalize it - we transform it into a standardized form that we can process.

We convert all capital letters to lowercase. We do this because all the words in n-grams are only lowercase, which simplifies the work and saves a lot of memory (if we process data where both uppercase and lowercase letters appear, processing and searching such data would be much more complicated). We get the neighborhood word. When comparing, we use only 4-grams. If the first word in a 4-gram is equal to the searched word, we need three following words after our word for comparison. If the last word in the 4-gram is equal to our word, we need three words before our word for comparison. Therefore, we consider the word context three words before and three words after this word, of course, if these words exist.

Subsequently, we find diacritics form of the word (more information will be provided in the next section, see Figure 2). If we successfully added the correct diacritical marks, we change the case of the letters in the new word to the original and write it into the resulting text. However, if we were not successful, we add the original word to the resulting text, and we do not need to change it to lowercase. When we insert words into the resulting text, we need to carefully insert punctuation, white and other characters to preserve the original format of the text. Consequently, if we have another word, we begin processing it, if not, our algorithm ends.

In the algorithm for restoring the diacritics form of a word using its surroundings (see Figure 2) we assume that we have a word for the restoration of the diacritics and its surroundings. First, we find all the n-grams for a word being processed. By the corresponding n-grams, we mean every n-gram in which at least one of the n words corresponds to our reconstructed word (of course, if we remove the diacritics from the n-gram before comparing them). If we do not find such n-gram, we end with a failure; this means that we have not added any diacritics to that word. If we find at least one such n-gram, we can proceed to their gradual processing. In the diagram (Figure 2), we see the for-loop for (n = 4; n > 0; n–), which shows that we gradually process groups of the corresponding n-grams, ranging from 4-grams to 1-grams. If there is no n-gram in the current group, we decrement n. In this way, we iterate from the group of n-grams to the group of (n-1) -grams while n > 0.

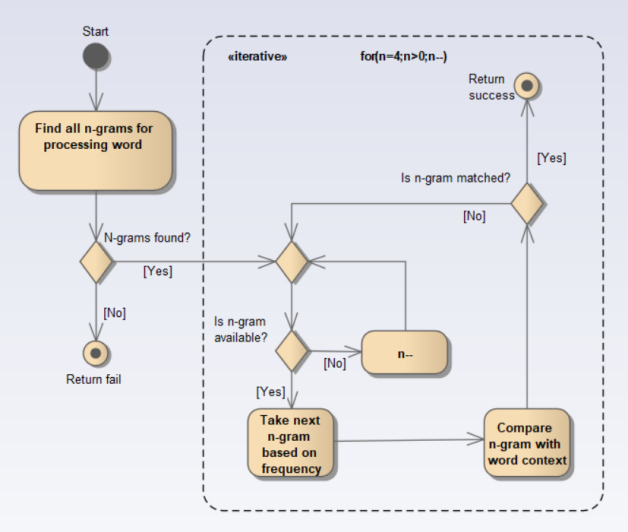

Figure 2: Activity diagram – matching n-grams by the surrounding context

If we have some n-gram in the current group, we take the one that has the highest absolute frequency. In the n-gram, we search where our word occurs (we have to keep in mind that the searched word may appear multiple times in the n-gram), and we compare the surrounding words of the n-gram with the surroundings of the reconstructed word (of course, we remove the diacritics from the n-gram before processing).

If we find a match, that is, the surroundings of the word agrees with the n-gram, we can return success and reconstruct diacritics by the found word from the n-gram. If not, we check if we still have any n-gram in the current n-gram group. We can notice the fact that if we go into the for-loop, the only possible exit is a success. This is because we assume that in the group of n-grams belonging to a given word there is necessarily at least one 1-gram. This condition ensures that if the loop goes to 1-gram comparison, immediately at the first 1-gram, the loop returns success because 1-gram has no neighborhood, and hence the match is definitive.

If we find a match around a word with an n-gram with a higher n, we consider it as more relevant than a match in a lower n, even if that n-gram had a higher absolute frequency. That's why we first search all 4-grams by frequency and then all 3-grams by frequency.

To help us quickly search through all the data associated with each word, we used the Trie data structure, which has an $O(n)$ search complexity where n is the number of items.

In order to quickly search all the data associated with each word, we need very fast memory access everywhere, based on the key that is the word. The Trie data structure (Retrieval, Prefix Tree, or Digital Tree) is perfect for this case. It is an ordered tree data structure, where the keys are usually strings (as in our case). Individual vertices are represented by parts of the key (in our case they are separate characters - letters). The tree trunk is an empty string and the leaves of the tree represent the data we store. For better clarity, let's look at the following Figure 3. The Trie data structure has the excellent complexity $O(n)$ to find the element, where n equals to the number of characters in the string. We take full advantage of this data structure to find n-grams for the word.

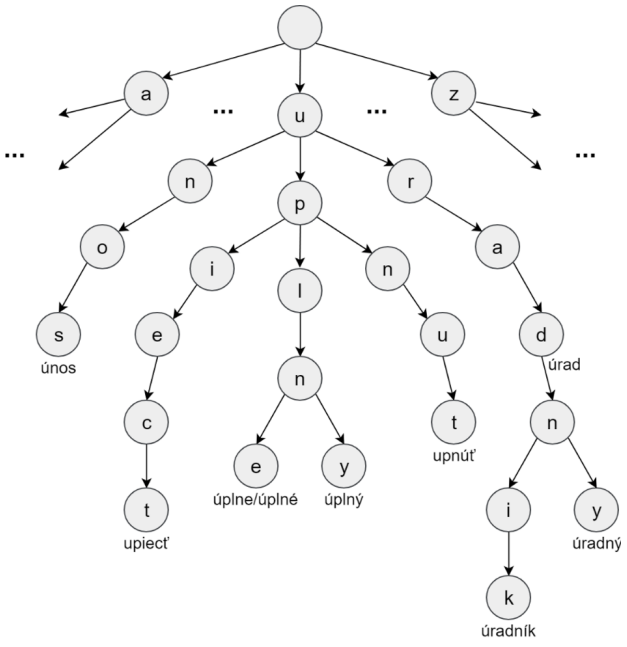

Figure 3: Example of data structure Trie with selected Slovak words

Because a compromise needs to be made between the speed and accuracy (success) of the algorithm, we decided to implement two different approaches: one focusing on speed (data structure in memory) and the other focusing on success (data structure on disk). It is always crucial how and where we store the data (n-grams) that we want to search.

3.1 The data structure in memory

Implementation of the data structure in main memory is very simple. We follow the algorithm design as shown in Figure 2. When we *find all n-grams for processing word*, the above-mentioned Trie structure comes into play. We chose the easiest approach – each word has its corresponding

n-grams at the top of the Trie tree. This ensures almost instant access to all necessary n-grams. Besides, the format of stored data in vertices can be adjusted. We do not have to perform a for-loop at all, nor do we search for the next n-gram from the *Take next n-gram based on frequency*. This means that all relevant 4-grams, then 3-grams, 2-grams and finally 1-grams will be on the first leaf. If we have n-grams stored in this memory, we don't have to hold information about their frequencies at all (which saves the memory). In result, the algorithm for finding n-gram is shown in Figure 4.

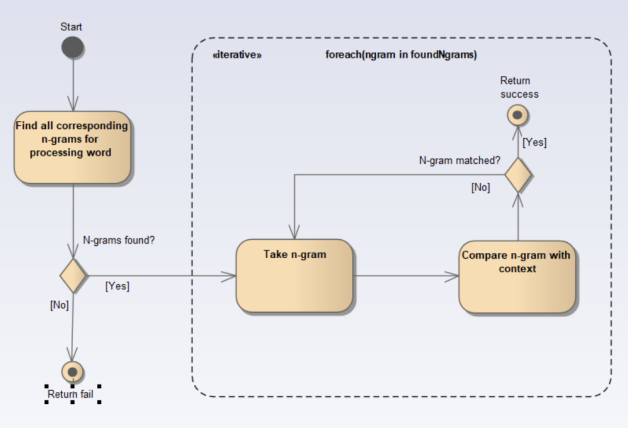

Figure 4: Activity diagram – n-gram diacritic search using Trie

Based on our observations and experiments, we can optimize the algorithm as follows:

1. There is no need to keep words without diacritics. They will be not found in the structure and skipped. Examples of such Slovak words are "roh", "lampa", "obloha" (in English: "the corner", "a lamp", "the sky").
2. If there is exactly one word with diacritics for one word without diacritics, we can only store it in 1-gram database and remove all n-grams containing the word. Examples of such Slovak words are: "akcionári", "vzájomnom", "ovzdušie", "veľvyslanec", etc. (in English: "shareholders", "mutual", "the atmosphere", "a ambassador").
3. 1-gram cannot have multiple diacritics variations, because there is no context to use for the match. So, 1-gram contains only one the most common form.

Thanks to these optimizations, the size and speed of the Trie data structure can be optimized without any loss of diacritic restoration.

We found that the number of used n-grams is always approaching the ratios showed in Table 2. The table shows the percentage of n-grams in a test database that was used in diacritics reconstruction. Using the memory only makes searching and the whole algorithm very fast, but we allow to run it on common PCs (average 8 GB RAM). We estimated that we must not exceed 4-5 GB of RAM to be able use such PC. To approximately reach this limit, the maximum number of n-grams must be limited to 700 per one word. Limit of 700 n-grams involves all n-grams levels with percentage distribution described in Table 2.

Table 2: Percentage amount of n-grams

	4-grams	3-grams	2-grams	1-grams
Percentage use of diacritics reconstruction	10%	24%	37%	29%

3.2 The data structure on disk

The memory restriction we set requires to use of a disk to store data of n-grams. Adding this possibility greatly increases the size of the potential database. Using disk as a storage for n-grams requires minor changes in our structure. Trie nodes contain the only reference to a binary file stored on disk as shown in Figure 5. More in detail, nodes in Trie will have one reference to index in the binary file,

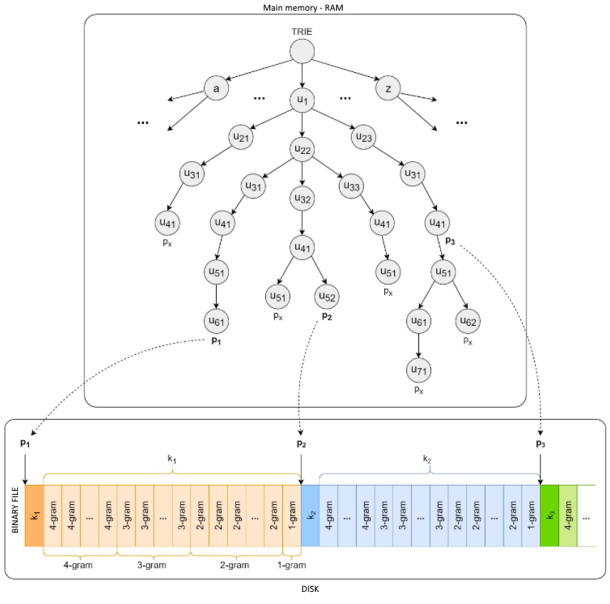

Figure 5: The structure of n-gram database using the binary file

the position p. This position in the file contains the count of n-gram k, which is a 4-byte number (an integer). Next, all n-grams of p position are processed. N-grams in the binary file are also descending ordered by occurrence frequency and grouped from 4-grams to 1-grams.

Since we have enough disk space, we can save the entire file without any problems. However, during our early tests, we found out that the reconstruction of diacritics was too slow. The reason was the frequently occurring words, which sometimes had several million n-grams that had to be compared. After thorough study of the words and their n-grams, we found that the number of n-grams per word (if we neglect the extremes of over-frequented words and unfrequented words) is most often in the range of 30,000-40,000. Therefore, we set an upper limit of 40,000 for the maximum number of n-grams per word (similar to the upper limit of 700 for Trie in RAM). We divided this threshold in the ratio of the use of n-grams groups according our experiments. We found that out of all words from test texts reconstructed with n-grams greater than 1 is distributed as follows: 15% of words were reconstructed using 4-grams, 35% using 3-grams and 50% using 2-grams. The following table (Table 3) shows the maximum number of n-grams stored in the structure by ratio of the distribution of the maximum number of n-grams.

Table 3: Maximum numbers of n-gram groups in a binary file

N-grams	4-grams	3-grams	2-grams	1-grams
The ratio of the distribution	15%	35%	50%	-
Max. count of stored n-grams	6,000	14,000	20,000	1

Using this limitation, we accelerated the reconstruction of one word from 0.27s to 0.009s, while we did not reduce the accuracy of reconstruction very much. All results of the final testing will be presented in the penultimate chapter.

To make our algorithm even faster without reducing the accuracy of the diacritic reconstruction, we introduced a cache memory into the algorithm. Let us assume that within the same text, the same words often appear in the same or similar context. Therefore, it would be advisable to store the n-grams we have found in some buffers or cache, so that we can access them more quickly when we encounter these words again. This cache must be limited because if we wanted to reconstruct too much text, the operating memory would quickly fill up and this could cause the application to crash due to insufficient memory. To achieve fast cache memory, we will use the Trie data structure along with a simple list that will serve as an auxiliary object of the Trie structure to direct it. We limit the cache to the last 1000 searched words with their appropriate n-grams. We will keep these words in that list. The algorithm will now work as shown in Figure 6.

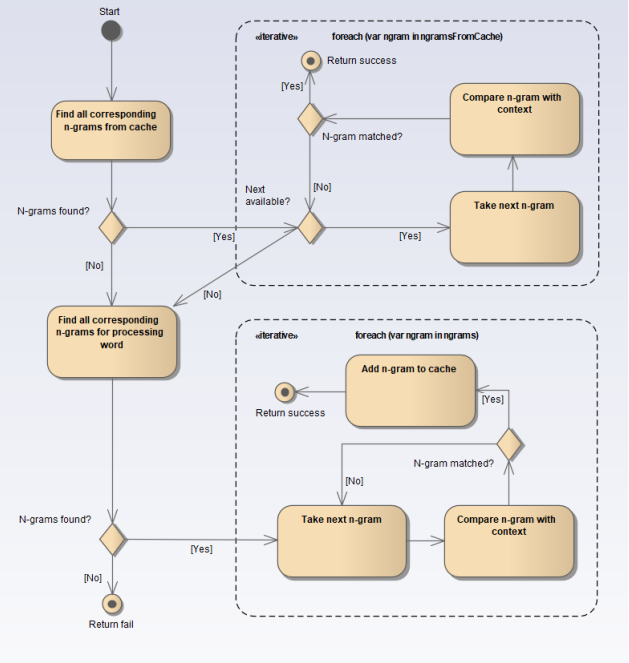

Figure 6: Activity diagram - implementation of diacritics restoration with cache memory

In the beginning, we look into the cache to see if it contains any corresponding n-grams for the current reconstructed word. If yes, we try to compare the equivalence of n-grams from the cache and the surroundings of the word. If we find the n-gram identical to the context, we will return success, otherwise, we continue with the above-mentioned procedure. However, after finding the same n-gram, we have one more step to put the n-gram into the cache, checking that the cache size has not exceeded its upper limit. If yes, we must remove the word with the n-grams that came first (FIFO principle).

The average reconstruction time of one word has been reduced by half compared to the last modification, from 0.009 s (9 ms) to 0.0046 s (4.6 ms). The accuracy of the algorithm was not affected.

4 Data

To build a good n-gram database, it is essential to use quality sources. Our data was created from public-open texts from the Slovak national corpus of Slovak Academy of Sciences [12]. The corpus contains a primary large number of words from a huge range of categories (many literary genres, regions, fields of science, etc.) from the year 1995. All words in the corpus contain additional information about them generated by *IRSTLM Toolkit* [1]. We have used a version of the corpus labelled as prim-8.0 released at 31.01.2018. For our algorithm, we are using sub-corpus prim-8.0-public-all, which contains all publicly accessible texts (71.10% journalistic, 15.22% artistic, 8.51% professional, 5.17% other texts). We chose four files from the sub-corpus:

- prim-8.0-public-all-word_frequency_non_case_sensitive.txt (containing non case sensitive tokens with frequencies, size 64.04 MB)
- prim-8.0-public-all-2-gramy.txt (containing 2-grams, size 2.99 GB)
- prim-8.0-public-all-3-gramy.txt (containing 3-grams, size 10.0 GB)
- prim-8.0-public-all-4-gramy.txt (containing 4-grams, size: 15.0 GB)

Since these text files also contain misspelled words and incorrect n-grams, they needed cleaning.

First, we removed words with irrelevant characters such as dot, comma, colon, quotes, or parentheses. Such characters will not be necessary for the reconstruction of diacritics. Next, we removed numbers, unknown characters, and n-grams with rare absolute occurrences. We also removed too long words, which were basically not correct. Since the files contained words with characters outside the Slovak alphabet, we defined a set of allowed characters. We included all characters from the Slovak alphabet and characters from languages used in neighboring countries of Slovakia, such as the Czech and German alphabet. In a deeper study of n-grams, we found out that they contain a lot of Czech and German expressions often used in Slovak. Such n-grams are several at 10,000 and have very high absolute frequencies.

After cleaning, we significantly reduced the set of n-grams as shown in the Table 4 and Figure 7. We used the output data to train our algorithm.

Table 4: The percentage utilization of original files

Text files	Original n-grams count	Selected n-grams count	Percentage utilization
1-grams	5,033,773	4,715,688	93.68%
2-grams	125,298,237	42,024,522	33.54%
3-grams	349,836,625	34,513,831	9.87%
4-grams	440,534,052	12,075,312	2.74%

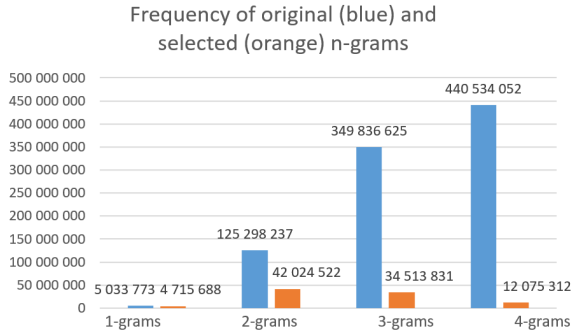

Figure 7: Comparison of original and filtered n-gram counts

5 Experiments

To verify the accuracy of the proposed algorithms, we chose a set of our test texts. We divided them into three categories according to literary styles: the professional, journalistic and artistic texts. After randomly selecting 25 texts by category, we obtained a set of more than 53,000 words, each category having approximately 17,000-18,000 words.

Subsequently, we removed diacritics from these texts and performed tests with our and mentioned algorithms. After that, we compared the reconstructed words with the original words. The overall results of the tests show Table 5 and Figure 8.

Also, we did another experiment in which we used part of the dataset from the paper by Náplava et al. [10] published on [12]. Specifically, we chose a test set for the Slovak

Table 5: Results of algorithms on our dataset sorted by success rate.

Algorithm	Error rate	Success rate
Our algorithm (disk)	1.83%	98.17%
Our algorithm (memory)	1.93%	98.07%
BRM	3.00%	97.00%
SAS	3.05%	96.95%
STUBA	4.53%	95.47%

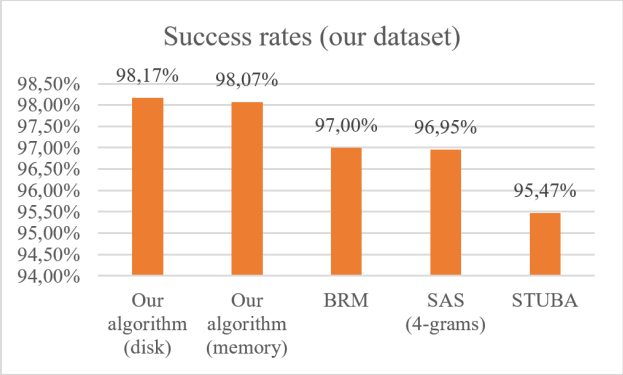

Figure 8: The chart of overall success rates of the algorithms on our dataset

language obtained from Wikipedia articles that contains 524,694 words.

Since individual web diacritics restoration algorithm services have limitations on the length of input text, we have to divide the test part of the dataset into subsets with words of max. 9,000 characters. We reached 369 subsets, but only 350 of them were used because we had trouble getting the results from the STUBA algorithm. The reasons were unhandled exceptions and errors with empty results on 19 subsets of texts. Nevertheless, we further found that this reconstructor also changed the content of the resulting texts, so it was not possible to reliably compare its accuracy based on words. For this reason, we did not include its results in the overall results. Besides, in the case of SAS, we also performed experiments for the 6-gram algorithm as shown in Table 6 and Figure 9.

Unfortunately, we did not have the opportunity to carry out our tests with data using a neural network as described in [10], where the success rate for the Slovak language reached 98.84% and using the language model even 99.09%. This shows that the neural network can achieve the best results at present. Despite this, our proposed algorithm achieves better results compared to other algorithms freely available and accessible on the web.

Table 6: Results of algorithms on Wikipedia dataset sorted by success rate.

Algorithm	Error rate	Success rate
Our algorithm (disk)	1.33%	98.67%
Our algorithm (memory)	1.53%	98.47%
BRM	1.70%	98.30%
SAS (6-grams)	2.38%	97.62%
SAS (4-grams)	2.47%	97.53%

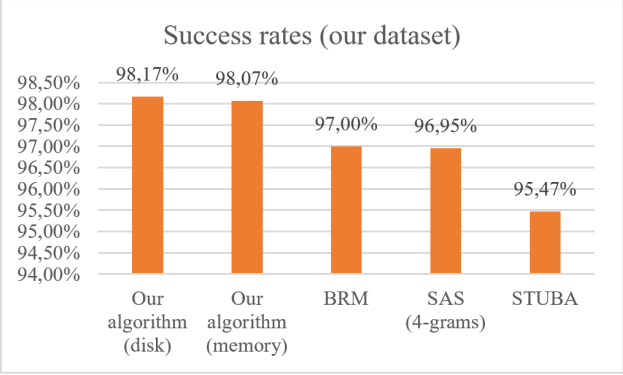

Figure 9: The chart of overall success rates of the algorithms on Wikipedia dataset

5.1 Our algorithm in memory

The first algorithm with the data structure in the main memory reached a total success rate of 98.07%, using n-grams diacritics for 83.03% of words were reconstructed. The maximum size of data in the main memory was 4.28 GB. The average speed of the reconstruction of one word was 0.24 ms. Table 7 shows the overall results.

Table 7: Test results – Statistics for our algorithm in memory (our dataset).

	Professional	Journalistic	Artistic	Total
Words in text	18437	17672	17187	53296
Use of 4-grams	373	576	198	1147
Use of 3-grams	2014	2884	1687	6585
Use of 2-grams	6475	7591	8027	22093
Use of 1-grams	5197	3917	5315	14429
Use of all n-grams	14059	14968	15227	44254
Time (ms)	3994	4363	4301	12658
Error words	25	223	513	1026
Success rate	99.86%	98.74%	97.02%	**98.07%**

5.2 Our algorithm on disk

The results of the second algorithm with data structure on the disk are shown in Table 8. This algorithm has a better success rate than the previous but is slower. It used up to 44.56% of all n-grams from the cache. The maximum size of data in RAM was 1.16 GB and the average speed of the reconstruction of one word was 4.17 ms.

Table 8: Test results – Statistics for our algorithm on disk (our dataset).

	Professional	Journalistic	Artistic	Total
Words in text	18437	17672	17187	53296
Use of 4-grams	835	1476	559	2870
Use of 3-grams	3054	4538	3061	10653
Use of 2-grams	5338	5915	6091	17344
Use of 1-grams	4832	3039	5516	13387
Use of all n-grams	14059	14968	15227	44254
Words from cache	6402	6518	6801	19721
Words from cache	45.54%	43.55%	44.66%	44.56%
Time (ms)	61708	79790	76311	217809
Error words	26	200	508	975
Success rate	99.86%	98.87%	97.04%	**98.17%**

5.3 SAS

The Ľ. Štúr Institute of Linguistics of the Slovak Academy of Sciences has 10 methods for reconstruction diacritics: first (selection of the first word), random (selection of random word), naive (selection of the most frequent word), n-gram (selection of word based on n-grams where n can be 2, 3, 4, 5 or 6), surreal and maximalist. The default method is 4-grams.

We found a few shortcomings of the reconstructor:

- Ignoring of multiple spaces – if there is more than one white space or white space between words, the SAS ignores them and uses only one.
- Adding diacritics to the Slovak word “a” (in English: “and”) – the word “a” without diacritics is the most common in the Slovak language. Nevertheless, the reconstructor very often added the accent "à", "á" or "ä".
- URLs – accents do not usually appear in URLs, but they are still added.

5.4 BRM

Although the BRM works only on adding the most likely word principle (or the highest occurrence), it achieves a very good success rate. It does not contain issues described in the previous text.

5.5 STUBA

Using our test texts and texts from Wikipedia we uncover some errors of this diacritics reconstructor:

- It ignores multiple spaces and they are replaced by a single space.
- Upper and lower-case letters are not formatted as in the input text. This makes a word as names and titles incorrectly written, such as MP3 -> mp3.
- Issues with quotation marks - quotations marks can be presented with multiple symbols (" „ “ ” ’ ‘ ‘ ’ … :). The words with these symbols were missing.
- Numbering – if a word contains a number, this number is replaced by various characters.
- Words “ty”, “TY”, “Ty” and “tY” throw an exception and cause the application crash.

6 Conclusion

In this work, we dealt with the problem of diacritics reconstruction in texts without diacritics. We started by describing the problems of texts without diacritics. We have analysed the reasons why such texts were created. Then we dealt with the methods of diacritics reconstruction. We described some available tools that can be used for this purpose. Then we proposed our own approach for diacritics restoration. We tested the algorithms, collected the results and compared them with existing tools. Compared to other available algorithms and services, our proposed algorithm achieved the best results and the lowest error rate. In addition to the algorithms described, there are others that use artificial intelligence. These tend to achieve greater accuracy. In the future, we will focus on improving the algorithm using various types of neural networks. For this purpose, we consider to use a bidirectional LSTM (long short-term memory) neural network with attention blocks.

References

[1] Federico M., Bertoldi N., Cettolo M., Irstlm: an open source toolkit for handling large scale language models, Ninth Annual Conference of the International Speech Communication Association, 2008.

[2] Gedera J., Doplňovač diakritiky (tool for diacritic restoration), http://text.fiit.stuba.sk:8081/, Last accessed 24 June 2020.

[3] Hládek D., Staš J., Juhár J., Diacritics restoration in the slovak texts using hidden markov model, Language and Technology Conference, Springer, 2013, 29–40.

[4] Hraška R., Doplňač diakritiky (tool for diacritic restoration), https://diakritika.brm.sk/, Last accessed 24 June 2020.

[5] Hucko A. Lacko P., Diacritics restoration using deep neural networks, 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA), IEEE, 2018, 195–200.

[6] Jansen W. Delaitre A., Mobile forensic reference materials: A methodology and reification, 2009, US Department of Commerce, National Institute of Standards and Technology.
[7] Krchnavy R. Simko M., Sentiment analysis of social network posts in slovak language, 2017 12th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP), IEEE, 2017, 20–25.
[8] Ljubešić N., Erjavec T., Fišer D., Corpus-based diacritic restoration for south slavic languages, Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16), 2016, 3612–3616.
[9] Merriam-Webster, Diacritic - definition of diacritic by merriam-webster, https://www.merriam-webster.com/dictionary/diacritic, Last accessed 24 June 2020.
[10] Náplava J., Straka M., Straňák P., Hajic J., Diacritics restoration using neural networks, Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), 2018.
[11] Novák A. Siklósi B., Automatic diacritics restoration for hungarian, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, 2286–2291.
[12] Náplava J., Straka M., Hajič J., Straňák P., Corpus for training and evaluating diacritics restoration systems, URL http://hdl.handle.net/11234/1-2607.
[13] Tufiş D., Ceauşu A., et al., Diac+: A professional diacritics recovering system, Proceedings of LREC 2008, 2008, 167–174.
[14] L'udovít Štúr Institute of Linguistics of the Slovak Academy of Sciences (JÚĽŠ SAV), Diakritik – nástroj na rekonštrukciu diakritiky (tool for diacritics reconstruction), https://www.juls.savba.sk/diakritik.html, Last accessed 24 June 2020.
[15] L'udovít Štúr Institute of Linguistics of the Slovak Academy of Sciences (JÚĽŠ SAV), Pravidlá slovenského pravopisu. 3., upravené a doplnené vyd, Bratislava: Veda, 2000.
[16] Zitouni I., Sorensen J. S., Sarikaya R., Maximum entropy based restoration of arabic diacritics, Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2006, 577–584.