Intelligent information systems

UDC 004.7 **doi:** https://doi.org/10.20998/2522-9052.2023.1.08

Anton Havrashenko, Olesia Barkovska

Kharkiv National University of Radio Electronics, Kharkiv, Ukraine

ANALYSIS OF TEXT AUGMENTATION ALGORITHMS IN ARTIFICIAL LANGUAGE MACHINE TRANSLATION SYSTEMS

Abstract. The work is **devoted** to the development of an organizational model of the machine translation system of artificial languages. The main **goal** is the analysis of text augmentation algorithms, which are significant elements of the developed machine translation system at the stage of improvement of new dictionaries created on the basis of already existing dictionaries. In the course of the work was developed a model of the machine translation system, created dictionaries based on texts and based on already existing dictionaries using augmentation **methods** such as back translation and crossover; improved dictionary based on algorithms of n-grams, Knuth-Morris-Pratt and word search in the text (such as binary search, tree search, sqrt decomposition). In addition, the work implements the possibility of using the prepared dictionary for translation. Obtained **results** can improve existing systems of machine translation of the text of artificial languages. **Practical significance** of this work is the analysis and improvement of text augmentation algorithms by changing the prefix tree type. Compared to the conventional algorithm, the improved algorithm reduced the memory usage by almost 13 times, which allows it to be used on much larger test data. This was achieved by changing the internal system of the node of the prefix tree from constant references to an expandable list.

Keywords: translation; augmentation; prefix tree; dictionary; artificial language.

Introduction

There are 2500-3000 languages on Earth. These languages differ both in prevalence and social functions, as well as in phonetic and vocabulary features, morphological and syntactic characteristics. In linguistics, there are a number of classifications of languages. The main ones are four: areal, genealogical, typological and functional.

Genealogical classification is based on the definition of family relationships between languages. At the same time, the common origin of related languages is proven and their development from a single language, often reconstructed in special ways, which is called the native language, is demonstrated. When genealogically classifying languages, first of all, the degree of their family relations and connections is clarified.

Typological (morphological), operates with classes of languages, combined according to those features that are selected as the most significant features that reflect the language structure (for example, the way morphemes are connected). The most famous morphological classification of languages, according to which languages are divided using the abstract concept of type into the following four classes:

– insulating or amorphous, for example, the Chinese language;

– agglutinative or agglutinative, for example Turkic and Bantu languages;

– incorporating, or polysynthetic, for example Chukotka-Kamchatka language;

– inflectional languages, for example Slavic, Baltic.

Areal (classification of languages is possible both within the genealogical classification of languages (for example, the Polish area, which includes Belarusian-Ukrainian dialects and slang [1, 2]), and for languages of different genetic affiliation (for example, the Carpathian area of Hungarian-Slavic dialects). In areal classification, an important role is played features related to contact phenomena.

Areal classification is also possible within one language in relation to its dialects, it is the basis of linguistic geography. Geographical classification is related to the place of distribution (primitive or late) of this or that language (or dialect). Its purpose is to determine the range of a language (or dialect) taking into account the boundaries of its linguistic features. The main method of research is linguo geographic. A special category of areal classification of languages is formed by linguistic conjunctions, which are formed as a result of linguistic interaction in the sphere of economic and household communication. Within the framework of a linguistic conjunction, convergence of related and unrelated languages included in it, etc dialects, which are combined by a certain commonality of economic and household vocabulary, syntactic constructions, characteristic features of morphology and phonetics. Thus, areal classification consists in the study of the linguistic map of the world, the linguistic characteristics of different countries, as well as the distribution of individual languages or groups of languages (Fig. 1).

Artificial languages are specialized languages in which vocabulary, phonetics, and grammar have been specially developed to fulfill certain goals. It is purposefulness that distinguishes artificial languages from natural ones. Sometimes these languages are called false languages. There are already more than a thousand such languages, and new ones are constantly being created [3].

The reasons for creating an artificial language are: facilitating human communication (international auxiliary languages, codes), providing fiction with additional realism, linguistic experiments, providing communication in a fictional world, language games and enjoyment.

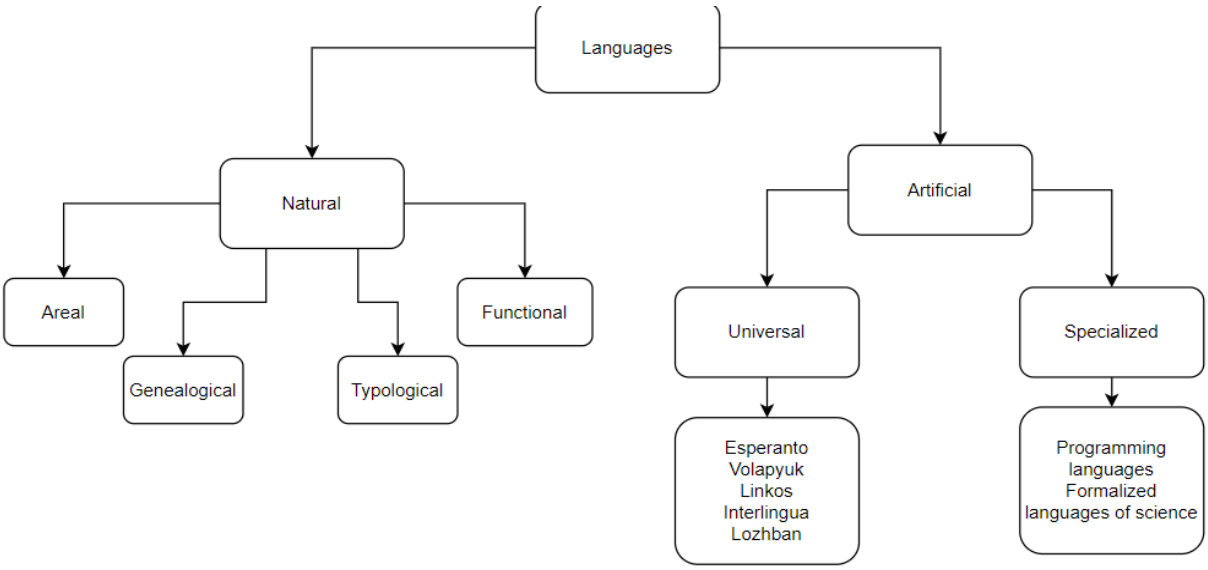


Fig. 1. Language types

The term "artificial language" is sometimes used to refer to planned languages and other languages designed for human communication. Sometimes they prefer to call such languages precisely "planned", since the word "artificial" can have a derogatory connotation in some languages.

Most artificial languages are created by one person, for example – Talos. But there are languages that were created by a group of people, such as Interlingua, developed by the International Assistive Language Association, and Lojban, created by the Logical Languages Group.

Analysis of the literature

Natural-language processing (NLP) is a general area of computer science, artificial intelligence and mathematical linguistics. It studies the problems of computer analysis and synthesis of natural language. In terms of artificial intelligence, analysis means understanding language, and synthesis means generating intelligent text. Solving these problems will mean creating a more convenient form of interaction between a computer and a person. There are 4 stages of development of natural language processing [4] (Fig. 2).

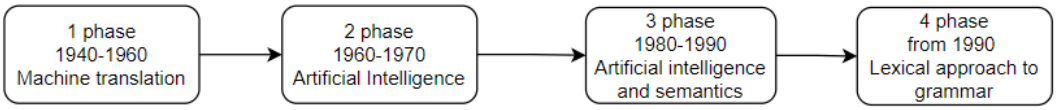


Fig. 2. NLP history

Understanding natural language is sometimes considered an AI-complete task because recognizing living speech requires a system's vast knowledge of its environment and the ability to interact with it. Defining the meaning of the word "understand" is one of the main tasks of artificial intelligence. Nowadays, ontologies, such as WordNet, UWN, play a significant role in solving natural language data processing problems. In the process of natural language processing research, significant results were achieved, including the development of powerful lexicographic systems [5, 6], programs for machine translation, electronic dictionaries, etc. However, there is a problem that still has not found its solution, it is rooted in the very nature of human language.

The problem of understanding human speech lies precisely in its ambiguity. Today, there are no programs that "understand" all types of ambiguities in a wide range of industries, but there are programs that can correctly respond to ambiguities in very narrow areas.

In NLP, algorithms are distinguished by basic algorithms (Fig. 3):

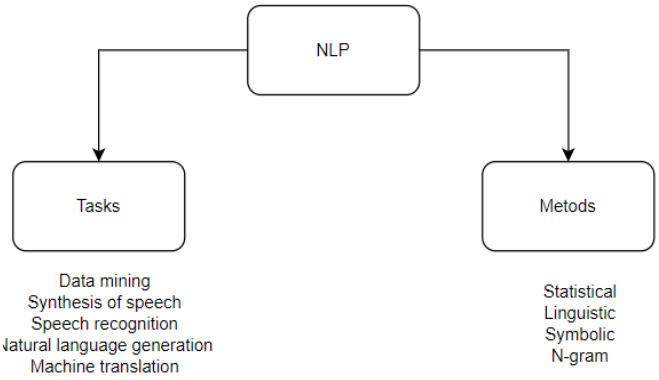


Fig. 3. NLP tasks and methods

– morphological analysis algorithms. They are used to recognize the elements or the morphological structure of the word - root, base, affixes, endings. Examples are stemming and lemmatization;

– lexical analysis algorithms. Lexical units of the text are used for recognition. The input of the algorithm is text, the output is a list of lexical units of the text.

Examples are lexical decomposition, which involves breaking the text into tokens; accordingly, programs that perform lexical decomposition are called tokenizers.

The problem formulation

The main goal of the project is the analysis of text augmentation algorithms in machine translation systems. To achieve the goal, the following tasks must be solved:

– development of a machine translation system model;

– creation of new dictionary based on the set of parallel text;

– creation of new dictionaries on the basis of already existing ones using augmentation methods [7];

– improvement of the dictionary thanks to the algorithms of n-grams and Knuth-Morris-Pratt [8-11];

– implementation of the possibility of using the prepared dictionary for translation;

– analysis of text augmentation algorithms.

Results and Discussion

A system with the following inputs, goals, and limitations was developed to accomplish the tasks.

Let's consider the application at a more detailed level.

The model consists of 4 main parts:

– a module for generating dictionaries based on input texts;

– a module for generating dictionaries based on already existing dictionaries of other languages;

– a module for updating probability of translation and improving dictionaries;

– translator program for translating texts with the help of dictionaries.

A generalized model of the proposed machine translation system is shown in the Fig. 4.

The detailed structure is shown in the Fig. 5.

Parallel texts are needed to generate dictionaries. These texts should have the same number of sentences. Sentences are counted by the number of period signs («.»), exclamation mark («!»), question mark («?»), three dots («...»).

In addition, each sentence must have at least one word. If, for example, there are only spaces, commas, quotation marks and other punctuation marks between the exclamation mark and the question mark, then it will not be considered a sentence.

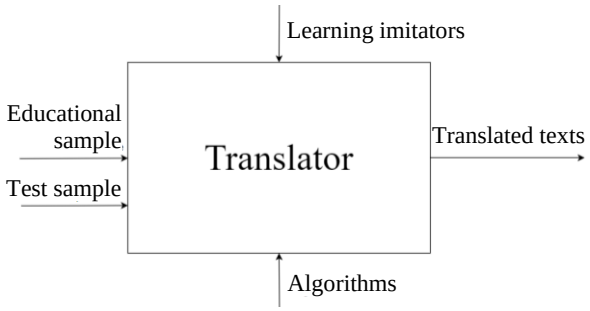


Fig. 4. A generalized model of the proposed machine translation system

In addition, sentences with the same serial numbers will be counted as translations of each other, that is, it is not possible to change the order of sentences for the correct operation of the program.

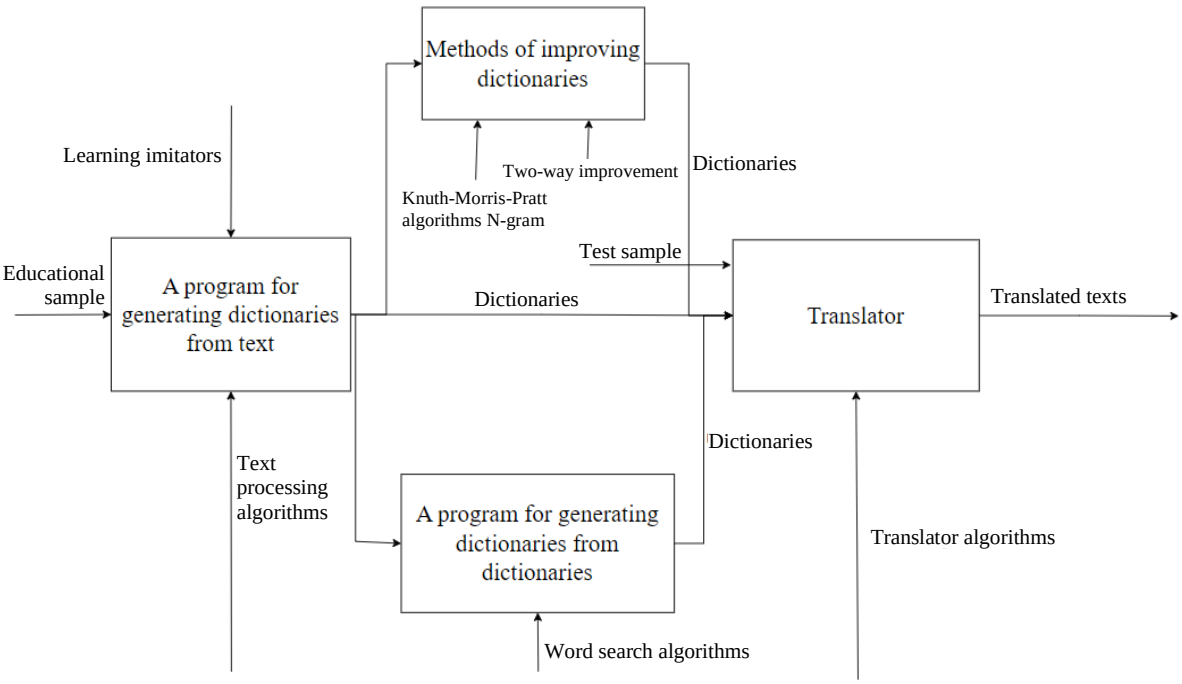


Fig. 5. Detailed model of the proposed machine translation system

To generate dictionaries from other dictionaries using the augmentation method, you need to perform the following procedure. Suppose we have generated a dictionary from language 1 to language 2, and from language 2 to language 3. Then, for each word from language 1, we must find translations in the language 3.

For this, for each possible translation of this word into language 2, we will find possible translations into language 3 (Fig. 6).

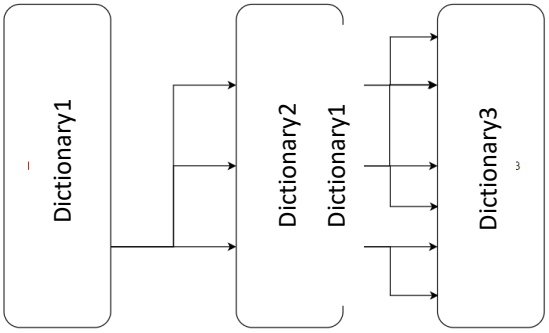


Fig. 6. Diagram of dictionary generation using an intermediate dictionary

Then the translation probabilities will depend on the individual probabilities of these translations. If the same word of language 3 occurs as a translation several times, its probability increases. Since this dictionary can then be used as a regular dictionary for translation, each of the obtained probabilities must be normalized in the range [0;1] (since the probability can increase due to repeated results, it may not be normalized depending on the calculation algorithm), and then multiplied by a factor to reduce the impact of the results of this algorithm.

Since the dictionary can be supplemented by conventional algorithms using texts, the results obtained using this method should affect the final result less than the results of the conventional algorithm. Since all results will be multiplied by this coefficient, it does not affect the translation result.

We refer to the improvement of dictionaries as a set of algorithms that are used depending on the input languages. This list consists of such modules as:

– N-gram;

– two-way improvement;

– addition of the finished dictionary with other texts;

– search for phraseological units;

– improvement based on morphological proximity.

As a result, we will get dictionaries of the following form (Fig. 7).

```
він : it 0.73333 is 0.68333 what 0.4 neither 0.3 world 0.26667  |
важко : it 0.8 is 0.74667 difficult 0.69333 to 0.64 imagine 0.58667  |
вам : you 0.78667 have 0.56988 been 0.53975 that 0.4795 the 0.35143  |
вам_вселили : you 0.8 have 0.75983 been 0.71967 that 0.63934 told 0.44762  |
ви : you 0.8 die 0.76522 to 0.73043 born 0.69565 should 0.69565  |
власна : our 0.8 about 0.6 own 0.6 what 0.4  |
вниз : along 0.74286 walk 0.71429 nothing 0.68571 if 0.65714 pushes 0.65714  |
вони : that 0.8 is 0.73846 they 0.73846 death 0.67692 are 0.67692  |
ворожий : hostile 0.8 nor 0.7 to 0.7 friendly 0.6 man 0.6  |
все : should 0.76522 and 0.73043 be 0.73043 die 0.69565 tormented 0.69565  |
вселили : have 0.8 you 0.75983 been 0.75983 that 0.6795 told 0.47619  |
вселили_що : have 0.8 you 0.75983 been 0.75983 that 0.6795 told 0.47619  |
два : two 0.8 these 0.64 images 0.64 are 0.48 incompatible 0.32  |
де : world 0.70476 where 0.70476 both 0.70476 are 0.70476 the 0.70476  |
див : of 0.8 miracle 0.68571 miracles 0.68571 a 0.57143 be 0.45714  |
дивом : a 0.8 be 0.68571 miracle 0.68571 would 0.57143 of 0.57143  |
для : to 0.66473 that 0.48618 you 0.47536 a 0.46222 is 0.40348  |
думкою : by 0.8 tormented 0.76522 this 0.76522 be 0.73043 thought 0.73043  |
живемо : live 0.8 told 0.77143 we 0.77143 been 0.74286 that 0.74286  |
життя : should 0.8 be 0.76522 tormented 0.73043 and 0.69565 by 0.69565  |
з : whis 0.8 a 0.68571 of 0.68571 be 0.57143 miracles 0.57143  |
задоволення : was 0.8 world 0.74667 created 0.74667 the 0.69333 solely 0.69333  |
значить : means 0.8 afraid 0.7619 it 0.7619 be 0.72381 of 0.72381  |
йдемо : go 0.8 an 0.77143 but 0.77143 of 0.74286 even 0.74286  |
кілька : some 0.8 to 0.74667 with 0.74667 you 0.69333 a 0.69333  |
краю : edge 0.68571 live 0.68571 on 0.68571 the 0.68571 of 0.68571  |
ласкавий : friendly 0.8 is 0.7 neither 0.7 it 0.6 nor 0.6  |
людині : man 0.8 hostile 0.7 to 0.7 nor 0.6 friendly 0.5  |
людина : man 0.8 been 0.76522 that 0.76522 have 0.73043 a 0.73043  |
ми : the 0.72381 we 0.68571 of 0.64762 on 0.45714 edge 0.45714  |
минуле : old 0.8 moral 0.74286 have 0.74286 values 0.68571 sunk 0.68571  |
монтерлан : monterlant 0.8 wrote 0.73846 meaningless 0.67692 are 0.61538 they 0.55385  |
```

Fig. 7. Dictionary example

Various methods can be used to save and search for text, such as:

– storage of words in the order in which they are presented;

– storage of words in alphabetical order;

– a balanced tree;

– hash table;

– root decomposition;

– prefix tree.

To improve the augmentation method in this work, we consider a prefix tree.

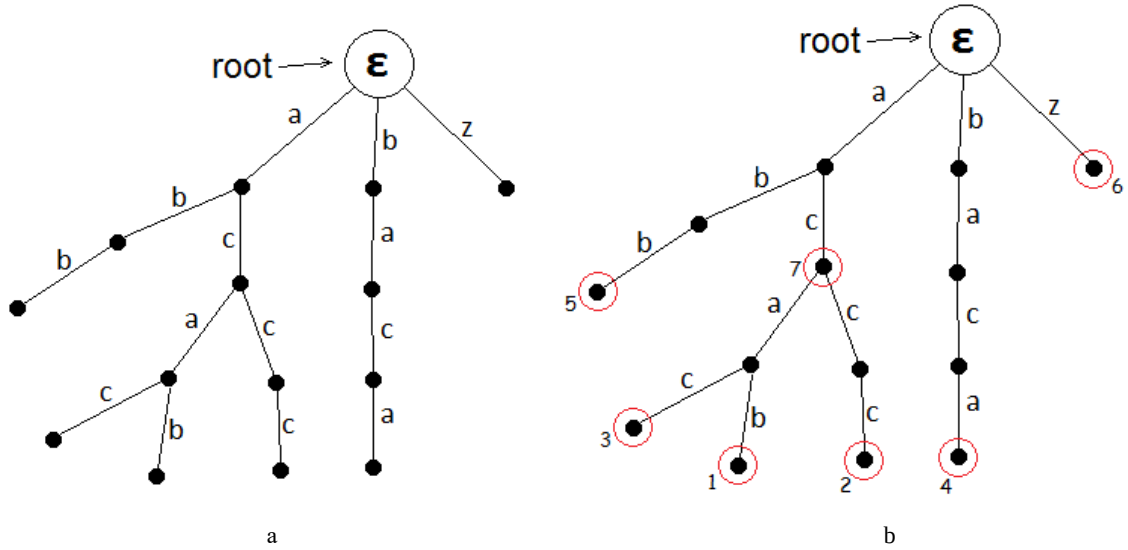
Strictly speaking, a prefix tree is a tree in which each node represents some string (the root represents the null string - ε). On the edges between the nodes, 1 letter is written, thus, going along the edges from the root to some nodes and targeting the letters from the edges in

the order of traversal, we will get a line corresponding to this node. The definition of a prefix tree as a tree also implies the unity of the path between the root and any node, therefore, exactly one row corresponds to each node (in the future, we will equate the nodes with the row it denotes).

We will build the prefix tree by sequentially adding the lines. At first, we have 1 node, the root is an empty string. Adding a line is done as follows: starting from the root, we move along our tree, each time choosing an edge that corresponds to the next letter of the line. If there is no such edge, we create it together with the node.

Here is an example of a constructed prefix tree for the strings:

1) “acab”,
2) “accc”,
3) “acac”,
4) “baca”,
5) “abb”,
6) “z”,
7) “ac”.



a b

Fig. 8. Example of a prefix tree a) usual tree b) a tree with labeled node

Note that when we add line 7, we don't need to add nodes at all, so we need to add a label if this node is the end of any line.

Obviously, searching for a word will be performed similarly to adding a new word. If the path does not exist, then instead of adding a node, we return the answer that the word does not exist. If at the end of the word there was no additional label in the last node, then we also return the same answer.

To test the effectiveness of the algorithm, we will use a method based on a previously known alphabet of 26 characters. We will use the following method to conduct the experiment algorithm. Let's generate the required number of long words (8-12 characters). Then we will generate the same number of words, and for each of the new words we will find it in the dictionary.

After the experiments, the following results were obtained (Table 1, Fig. 9).

Table 1 – **Prefix tree runtime results**

Size of input data, number of words	10^6	3*10^6	10^7	3*10^7	10^8
Operating time, sec	1	3	8	4*	12*

As it became known during the experiment, with a very large amount of input data, despite the fact that the algorithm works quite quickly, it requires quite large amounts of memory. This makes it impossible to work on a tree of large volumes.

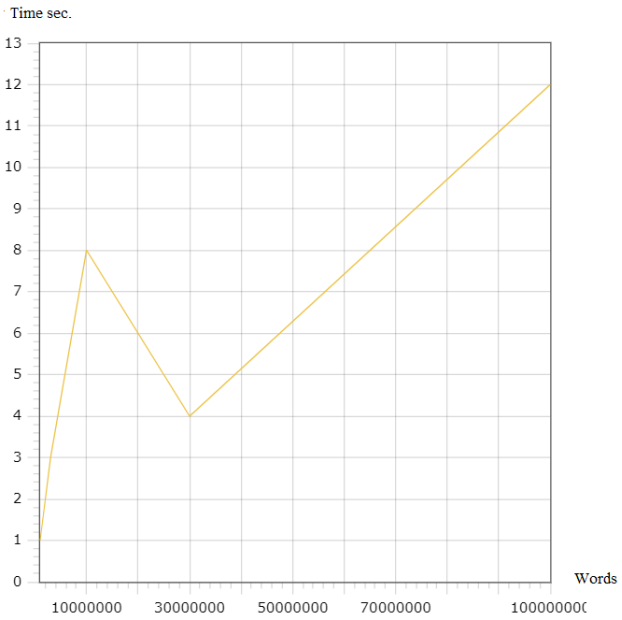


Fig. 9. Dependence of the execution time of the traditional prefix tree algorithm on the input data

Therefore, changes were made for the last two tests:

– instead of a tree built on 3*10^7 and 10^8 words, a tree was built from the first 10^6 words repeated 30 or 100 times, respectively;

– then they made the same 3*10^7 or 10^8 times requests to find the word.

Since the tree does not grow when entering the same word, the speed of work has decreased several times compared to the expected. Therefore, we can conclude that this version of the prefix tree has a limit close to 10^7 words when using the algorithm on a regular PC.

That is enough for ordinary languages.

Let's try to modify the algorithm so that it works under the necessary conditions. Since each node in the prefix tree keeps a reference to 26 other nodes, even if they don't exist, with a large amount of data, the standard implementation of the method does not meet our needs at all, or it requires computers that may have more memory available for work. However, since we don't know in which alphabet, we will use our program, things get much worse.

Therefore, we will try to make modifications to the operation of the algorithm, which will save memory regardless of time.

To save memory, we will change the structure of the nodes. If earlier we used static links to each letter, now we will use an expandable array, which will require us to spend more time accessing one letter, but will allow us to reduce the cost of the necessary memory.

After the experiments, the following results were obtained (Table 2, Fig. 10).

Table 2. Modified prefix tree runtime results

Size of input data, number of words	10^6	3*10^6	10^7	3*10^7	10^8
Operating time, sec	0	1	3	10	33

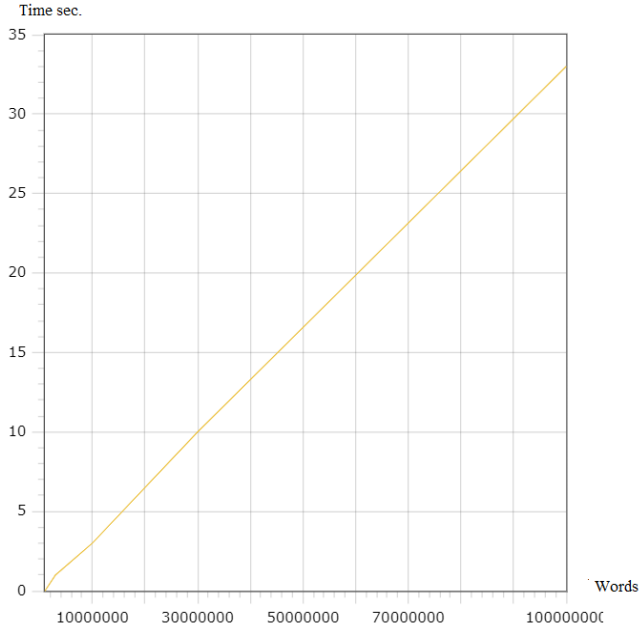


Fig. 10. Dependence of the execution time of the modified prefix tree algorithm on the input data

As we can see from the results, the time growth rate is linear with size of the input data. Therefore, on large data, this algorithm is slightly faster than binary search, which has logarithmic complexity. However, the time used to search for a letter is comparable to the logarithm, and will increase with the increase of the used alphabet, which can lead to a deterioration of the algorithm's results on words from a larger alphabet. Therefore, we cannot unequivocally say that this algorithm is better, it is necessary to pay attention to the data on which it will work, since it depends more on them compared to other algorithms.

You may also find it surprising that the algorithm that sacrifices time for the sake of memory began to work faster than the usual algorithm. There are two reasons for this. The first - since we make only one request for each word, the advantage of the usual algorithm, which responds to requests faster, is negated. Second, we have to allocate space for all 26 links, this takes some time. And so the time to build the tree begins to play a larger role in the overall running time of the program. Since we use generated words, one of the important quantities, such as the power of prefixes, grows as quickly as possible, which may not happen in ordinary text.

Let's understand what the power of prefixes is. Since we can see from the structure of the prefix tree (Fig. 7) that two words can contain common nodes, it is clear that the larger their common prefix (in this case, we consider the prefix not a morphological prefix, but a substring that begins with the beginning of the word), the tree will have less nodes.

For example, let's try to add the words “acca” and “eb” to the already considered prefix tree. Since the word “accc” already exists in the tree, adding this word will create one additional node. The word “eb” will create 2 new nodes, since there was no word starting with “e” yet.

Therefore, a longer word can produce fewer nodes than shorter.

In ordinary language, we have quite a lot of same rooted words, and words that begin with the same prefix, as languages were formed naturally. Therefore, in contrast to randomly generated sequences of letters that we consider a word, the prefix tree will show itself much better on ordinary words of a certain language. This is because the number of different prefixes in the generated sequence of letters will be much greater than the number of prefixes in a normal language. And since each nodes of the tree contains one prefix, as the power of the prefixes increases, the volume of the tree will also increase.

Conclusions

In the course of the work, an analysis of the machine text translation system for artificial languages and the augmentation methods used in it was carried out.

Improvements were made to the method of storing dictionaries using the prefix tree method, which was changed to save memory, which allowed it to be used on a home PC on large volumes of text.

Compared to the conventional algorithm, the improved algorithm reduced the memory usage by almost 13 times, which allows it to be used on much

larger test data. This was achieved by changing the internal system of the nodes of the prefix tree from constant references to an expandable list.

The model has quite a high potential for use in cases where there are no conventional translators. With a growing number of languages, this problem may worsen, as the number of translators cannot grow at the same rate.

REFERENCES

1. Manuel, K., Indukuri, K.V. and Krishna, P.R. (2010), "Analyzing Internet Slang for Sentiment Mining", *2010 Second Vaagdevi International Conference on Information Technology for Real World Problems*, pp. 9–11, doi: https://doi.org/10.1109/VCON.2010.9
2. Ren, F. and Matsumoto, K. (2016), "Semi-Automatic Creation of Youth Slang Corpus and Its Application to Affective Computing", *IEEE Transactions on Affective Computing*, April-June 2016, vol. 7, no. 2, pp. 176–189, doi: https://doi.org/10.1109/TAFFC.2015.2457915
3. Kazakov, D. (2017), "Artificial naturalness", Science and life, no. 10, pp. 100–107, available at: https://www.nkj.ru/archive/articles/32254/
4. Karen, S. Jones (2001), *Natural language processing: a historical review*, Cambridge: Computer Laboratory, University of Cambridge, available at: https://link.springer.com/chapter/10.1007/978-0-585-35958-8_1
5. Ryzhkova, V. (2020), "Possibilities of Computer Lexicography in Compiling Highly Specialized Terminological Printed and Electronic Dictionaries (Field of Aviation Engineering)", *Ivannikov Memorial Workshop (IVMEM) 2020*, pp. 40–42, doi: https://doi.org/10.1109/IVMEM51402.2020.00013
6. Ranaivo-Malançon, B., Saee, S. and Wilfred Busu, J.F. (2014), "Discovering linguistic knowledge by converting printed dictionaries of minority languages into machine readable dictionaries", *2014 International Conference on Asian Language Processing* (IALP), pp. 140–143, doi: https://doi.org/10.1109/IALP.2014.6973522
7. Chumarina, G.R. (2013), "Classification of electronic dictionaries in modern lexicography and lexicologists and features of their use", Baltic Humanitarian Journal, No. 4, pp. 123–126.
8. Anggreani, D., Putri, D.P.I., Handayani, A.N. and Azis, H. (2020), "Knuth Morris Pratt Algorithm in Enrekang-Indonesian Language Translator", *2020 4th International Conference on Vocational Education and Training* (ICOVET), 2020, pp. 144–148, doi: https://doi.org/10.1109 / ICOVET50258.2020.9230139
9. Zaiceva, S. and Barkovska, O. (2020), "Analysis of Accelerated Problem Solutions of Word Search in Texts", The Fourth International Scientific and Technical Conference «*Computer and information systems and technologies*», NURE Kharkiv, p. 66, doi: https://doi.org/10.30837/IVcsitic2020201445
10. Barkovska, Olesia, Mikhal, Oleg, Pyvovarova, Daria, Liashenko, Oleksii, Diachenko, Vladyslav and Volk, Maxim (2020), "Local Concurrency in Text Block Search Tasks", *International Journal of Emerging Trends in Engineering Research*, Vol. 8. No. 3, March 2020, pp. 690–694, doi: https://doi.org/10.30534/ijeter/2020/13832020
11. Barkovska, O., Pyvovarova, D., Serdechnyi, V. and Liashova, A. (2019), "Accelerated word-image search algorithm in text with adaptive decomposition of input data", *Control, Navigation and Communication Systems*, vol. 4 (56), pp. 28–34, doi: https://doi.org/10.26906/SUNZ.2019.4.028 (in Ukrainian)

Received (Надійшла) 14.12.2022
Accepted for publication (Прийнята до друку) 22.02.2023

ABOUT THE AUTHORS / ВІДОМОСТІ ПРО АВТОРІВ

Гаврашенко Антон Олегович – аспірант кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Anton Havrashenko – postgraduate student at of Electronic Computers Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: anton.havrashenko@nure.ua; ORCID ID: http://orcid.org/0000-0002-8802-0529.

Барковська Олеся Юріївна – кандидат технічних наук, доцент, доцент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Olesia Barkovska – Candidate of Technical Sciences, Associate Professor, Associate Professor of Electronic Computers Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: olesia.barkovska@nure.ua; ORCID ID: http://orcid.org/0000-0001-7496-4353.

Аналіз алгоритмів аугментації тексту в системах машинного перекладу штучних мов

А. О. Гаврашенко, О. Ю. Барковська

Анотація. Робота присвячена розробці організаційної моделі системи машинного перекладу штучних мов. Головною **метою** є аналіз алгоритмів аугментації тексту, які є значущими елементами розробленої системи машинного перекладу на етапі вдосконалення створених нових словників на основі вже існуючих словників. В ході виконання роботи була розроблена модель системи машинного перекладу, створені словники на основі текстів та на основі вже існуючих словників **методами** аугментації такими, як зворотній переклад та кросовер; вдосконалено створений словник на основі алгоритмів n-грамм, Кнута-Моріса-Пратта та пошуку слів у тексті (таких, як бінарний пошук, пошук в дереві, пошук в кореневій декомпозиції). Окрім того, в роботі реалізована можливість використання підготовленого словнику для перекладу. Отримані **результати** можуть покращити існуючі системи машинного перекладу тексту штучних мов. **Практичною значущістю** даної роботи є аналіз та покращення алгоритмів аугментації тексту за допомогою зміну типу префіксного дерева(бора).Порівняно зі звичайним алгоритмом, покращений алгоритм дозволив скоротити використання пам'яті майже в 13 разів, що дозволяє використовувати його на набагато більших тестових даних. Це було досягнуто завдяки зміні внутрішньої системи вершини бору із константних посилань, на розширюваний список.

Ключові слова: переклад, аугментація, префіксне дерево, словник, штучна мова.