Improvement of the algorithm of automated definition of rhyme

Vladimir Barakhnin
Federal Research Center for Information and Computational Technologies
Novosibirsk, Russia
bar@ict.nsc.ru

Olga Kozhemyakina
Federal Research Center for Information and Computational Technologies
Novosibirsk, Russia
ORCID: 0000-0003-3619-1120

Ilya Pastushkov
Federal Research Center for Information and Computational Technologies
Novosibirsk, Russia
ORCID: 0000-0002-0341-7931

Irina Kuznetsova
Federal Research Center for Information and Computational Technologies
Novosibirsk, Russia
ORCID: 0000-0002-6890-1636

Yulia Borzilova
Federal Research Center for Information and Computational Technologies
Novosibirsk, Russia
ORCID: 0000-0002-8265-9356

***Abstract*—The paper considers approaches to the improvement of one of the steps of the algorithm used for the automated determination of rhyme in poetic texts. The automated rhyme detection tool is one of the modules of the system of complex analysis of poetic texts. In the current module implementation, the rhyme search and definition subtask are solved by finding words with consonant endings using the A. A. Zaliznyak Grammar Dictionary of the Russian Language and the basic rules of phonetic analysis. Alternative solutions to the search problem in the dictionary of words with consonant endings are proposed. The results obtained will allow us to conclude that the current implementation is optimistic and the methods used can be finalized to solve the problems of determining the rhyme of a poetic text.**

Keywords—analysis of poetic texts, metrorhythmic analysis, rhyme identification

I. INTRODUCTION

One of the tasks of the automated complex analysis of poetic texts [1] is to determine the characteristics which are related with the metrorhythmics of a poem. Among the works where the statistical information extracted from the poetic text was used for the solving of philological problems, we can mention the study [2], which, despite compiling it manually, presents a rather comprehensive statistical picture of the metrorhythmics of Pushkin's works, what allows the authors to find the patterns inherent to Pushkin's rhyme. The modern information technologies make possible to conduct such studies, if not completely automatically, then with minimal usage of the work of expert-philologists.

The problem solution of automated analysis of poetic texts requires the adaptation to various languages. The different approaches are caused by both the specifics of the language (in particular, the features of the construction of poetic texts) and the tools used by researchers. The toolkit, in turn, depends on the goals set by the researchers (for example, to obtain the confirmation of any regularity in the structure of the poetic text), and, to some extent, on technologies that were relevant at the time of the study.

As for the linguistic versatility of instruments, it is impossible to develop a system for automated analysis of meter and rhythm, designed for a wide range of languages. Moreover, the insoluble task is to develop a metrorhythmic analysis system suitable for at least a group of related languages — each language requires the development of its own approaches that take into account its structure. The authors of the study [3] conducted an experiment for languages similar in structure, which ended unsuccessfully due to the specifics of each of the languages considered by the authors.

The problem of analyzing the metrorhythm of poetic texts for each language (or a group of the similar languages) is obtained differently. Next, we will consider some of the projects of the authors who solve the indicated problem for different languages.

D. Fusi in studies [4, 5] introduced the *Chiron* system, which allows analyze with several languages (Latin and Greek). The system is built at a level of abstraction in which it is possible to work with several different languages, meters and texts. The developed system have a modular structure, each module interacts with the next one by data transfering (in a predetermined format). The higher the level of the hierarchical chain, the more abstract analysis is performed by this component. Hierarchy levels in the system:

- phonetics and prosody;
- appositives and clique;
- metric scan.

The author [4, 5] does not mention the accuracy statistic in determining each level (phonetics, clitics, metrics), but it can be assumed that the accuracy is not the maximum. The author emphasized that the developed system (as well as similar ones) does not imply a complete replacement of the expert; the main task is to provide researchers with data whose processing costs occupy a significant share of human resources.

B. Navarro proposed a tool that studies the metrics of Spanish sonnets and performs semantic analysis of poems [6, 7]. Currently, this system is applied to a corpus of 5078 sonnets of the XVI and XVII centuries. The corpus is converted to the TEI format[1]; the sequence of characters from one poem without additional marking is input to the system. A rule-based module performs separating syllables: an external grammar marking system is used. If the syllabic partition produces 10 metric syllables, then the system considers that the scan is complete. For non-standard situations, a number of rules applied (a detailed description of the rules is not given by the authors of the project).

[1] TEI: Text Encoding Initiative: https://tei-c.org/

The system [3], mentioned previously, is a tool for the complete analysis of poetic texts in Portuguese. The system input is a poems in XML format and it scan each poem independent of other poems in the corpus. Includes the following steps:

- text preprocessing (conversion to XML format);
- extract words from a poem;
- finding a stressed syllable;
- division into syllables;
- phonetic transcription forming (using an independent dictionary);
- selection of transcription options for each poem (determination of the rhythmic scheme);
- an attempt to determine the metric of a poem;
- search for matching metrics based on the most appropriate rhythmic scheme;
- splitting a work into syllables according to metric.

The results of the analysis [3] showed a high percentage of accuracy (95–98%), however, for other languages (similar in structure), the experiment on the analysis of poetic texts ended unsuccessfully due to the specifics of individual languages.

M. Agirrezabal et al. [8] developed the *ZeuScansion* system, which performs syntax analysis for English poems. The system uses dictionaries to determine the stressed syllable in a word. By combining words to form the stress pattern of the whole poem, the system also performs syntax analysis, followed by a series of rules. If the word was not found in the dictionary, the program searches and uses the nearest word in heuristics.

R. Ibrahim and P. Plecháč [9, 10] developed the *KVĚTA* system, the purpose of which is to analyze Czech poems. The system got a poem as input, the words of which should contain morphological marking. *KVĚTA* applies a series of rules to poems that transform a poetic text into a phonetic transcription; if the rules cannot be applied, a dictionary is applying. The system compares the patterns found in the poems with the generated variations. Initially, the idea of a metric index was used [11]. Later, the authors used a metric coefficient using some others parameters, which allowed to increase the accuracy from 94.88% to 95.94%.

A number of works are known devoted to the analysis of versification for Arabic and similar languages. A. Kurt and M. Kara [12] proposed an algorithm for recognizing and analyzing poems written in a special, typical for eastern (Arabic, Persian, Turkish) poetry, versification system "arud". M. A. Alnagdawi described a method for finding poetic metrics using context-free grammars [13]. A. Almuhareb et al. [14] described some methods for defining poetic patterns in Arabic for extracting verses.

For the Russian language, a number of solutions during metrorhythmic analysis problem are also proposed. In study [15], the automatic procedures for specifying a poetic text — metrorhythmic marking and identification of a verse meter — were considered. The automation of metrorhythmic marking is achieved by using the following procedures [16]:

- line numbering of the poem;
- tokenization of words;
- accentuation of the poem;
- selection of rhymed lines;
- syllabic determination.

The authors[2] developed an open network resource, which is represented by the components: the problem-oriented "Poetology Thesaurus" and the "Block of Analysis and Specification" of the text objects. In the "analysis and specification" block, two sets of tasks are identified [15]: a specification of terminological articles of a thesaurus and a specification of a poem. The structure of the complex includes groups of solutions of the problems:

- metrorhythmic marking of the text;
- filling of the fields of the specification of the poem;
- meter identification.

Among the tools that execute metrorhythmic analysis, web resources [3, 4] are of interest. The first of them, *Rifmoved.ru*, is positioned as an supporting tool for the analysis of poetic text, which determines the stanza and the forms (sonnet, sextine). The algorithms were developed on the basis of the author's concept of program poetry analysis by V. Onufriev, however, a theoretical description of these algorithms was not found in scientific sources. The authors of the resource indicate that the work of the algorithms is designed to analyze poetic texts written in traditional forms, classical stanzas and sizes. This fact greatly limits the usage of the tool for large corpuses of texts of poets who are not related to classical literature.

The second resource, the *Neogranka.ru*, obviously, is an amateur web portal for determining the poetic size, generating new poems and selecting rhymes. When a user tries to determine the verse size, the service clarifies all controversial situations (accentuation options), what takes a lot of user time. There is also no theoretical description of the algorithms used in available sources.

It is important to note that almost all the algorithms mentioned above are aimed to study relatively small text corps covering the work of one or more authors, therefore, the speed of rhyme determination algorithms is not a critical parameter. However, in the research conducted at the Federal Research Center for Information and Computational Technologies (FRC ICT), it is planned to study the interdependence of the phonometric and lexical-thematic levels of poetic texts with the aim of identifying and measuring the relationships of semantic associations described on the basis of semantic fields with poetic sizes; the so-called textures that take into account the construction and metrorhythmics (a detailed statement of the problem described in [17]). One of the main difficulties in solving problem mensioned above is the need to analyze corpus of poetic texts of a large volume, as a result of which the task of optimizing rhyme search algorithms from the point of view of time spending becomes necessary. As usual, these algorithms use multiple queries to databases containing

[2] Wikipoetics: http://wikipoetics.ru/
[3] Rifmoved.ru. http://rifmoved.ru/analiz_stihov.htm
[4] Neogranka.ru. http://neogranka.ru/razmer_stiha.html

phonetic transcription of words, so we are faced with the task of optimizing such SQL queries. Note that this task is becoming actual in all areas of scientific research working with Big data: from business analytics to the analysis of Earth remote sensing data (for example, [18, 19]).

II. The problem statement

A web application has been developed at FRC ICT[5], which is used to analyze the structural level of Russian-language poetic texts. The algorithms are described in [20], they does not involve a work with complex cases of analysis of poetic texts, therefore, in [21], the implementation of the algorithms from [15] was proposed, what includes a more rigorous classification of poems by meter. But in the algorithm for determining the rhyme from [15], the authors use the web-based application "Big Rhyme Dictionary"[6]: the application receives a word, the output returns the full set of words rhyming with it (out of context). However, this approach takes a lot of time and resources, therefore, the rhyme search algorithm [21] is implemented for reasons of the possibility of rhyme creation: the lines rhyme if the last words in the line have the same position of the stressed syllable and the endings phonetically match.

To identify the phonetically matching endings, the data about endings from article [22] are used. The algorithm request a word into a table with words aggregated from A.A. Zaliznyak's dictionary [23], implemented in a standard way.

The purpose of the study is to find for alternative approaches to search for rhyming lines in the database and conduct a series of experiments to find out the most effective method for usage in the algorithm. The proposed solution:

1) To build a table with inverted rows sorted lexicographically.

2) To separate all words into sections by ranges after endings. In other words, the store endings (inverted) separately from the word (as metadata).

3) To add the trigram symbolic indexes to the original table with all the words.

4) To perform an experiment with the aim of find rhyming words using sections (search only endings) and trigram indexes.

5) To compare the performance of a section search option using indexes or a combination of these options.

To test the hypothesis about the effectiveness of application of the trigram symbolic indexes, it was decided to conduct an additional experiment to measure the performance of SQL queries using the indexes in the search module of the complex analysis of poetic texts. This module solves the problem of searching for low-level characteristics (for example, metrorhythmic statistics) and high-level characteristics (for example, genre-style affiliation). When a search query is done, SQL queries to the database are generated, some of which include a search by values of the varchar and text type. The execution time of such queries can be reduced by using the symbolic indexes.

[5] Analysis of poetic texts online. http://poem.ict.nsc.ru/

[6] Big Rhyme Dictionary. http://rifmovnik.ru/docs.htm

III. The research process

A. Data preparation

One of the options for the search implementation is the partition of the source table with words into sections. The version of the *PostgreSQL* database deployed on the FRC ICT server supports simple partitioning: the splitting of one large logical table into several small physical sections[7]. The benefits of the usage of sections:

- When queries or updates access a large percentage of a single partition, the performance can be improved by taking advantage of sequential scan of that partition instead of using an index and random access reads scattered across the whole table.
- The bulk loads and deletes can be accomplished by adding or removing partitions, if that requirement is planned into the partitioning design. *ALTER TABLE NO INHERIT* and *DROP TABLE* are both far faster than a bulk operation. These commands also entirely avoid the *VACUUM* overhead caused by a bulk *DELETE*.
- Seldom-used data can be migrated to cheaper and slower storage media.

PostgreSQL supports partitioning via range partitioning (for example, one might partition by date ranges) and list partitioning − the table is partitioned by explicitly listing which key values appear in each partition. In this study, the list partitioning is used, where the ends of the dictionary words are indicated as key values.

To create a list of sections in form of tables, a *Python* script is used that operates by the following algorithm:

1) The request to a table with words.

2) The selection of the *N*-last characters from the word.

3) The formation of an array of all dictionary endings.

4) The counting and sorting the usage of each ending in descending order.

5) The separating of *M*-first endings from the array, on the basis of which the sectioning will be performed.

As the last *N* characters, four characters are taken, this value can be changed in the future. To build a sorted dictionary, we use the *collections* module of the *Counter* library[8]. The result is a dictionary of the following structure:

Counter({'НОГО': 86077, 'ЕЙСЯ': 76978, 'ВШЕЙ': 76400, 'ИМСЯ': 62934, 'ШЕГО': 61719, 'ГОСЯ': 57630, 'ИХСЯ': 57617, 'НОМУ': 57354, 'ВШИМ': 57282 ...})

In the received dictionary, the key is the desired ending (last *N* characters), and the value is the number of occurrences of this ending. It was decided to isolate the values of endings with coefficients included in the 90th percentile from the created dictionary. These endings were used to create the sections.

The process of the creation of partitioned tables includes the following steps:

[7] PostgreSQL: Documentation 9.4: Partitioning. https://www.postgresql.org/docs/9.4/ddl-partitioning.html

[8] Collections — Container datatypes. https://docs.python.org/3.7/library/collections.html

1) To create a parent table whose properties inherit all the child tables (sections). The parent table is a table with words from the dictionary of A. A. Zaliznyak [23] with the structure:

a) identifier;

b) word;

c) accentuation (the number of the syllable to which the stress falls);

d) word type;

As an additional column, the endings (last *N* characters) of each word in inverted order are added.

2) To create the child tables with inheritance of parent structure. In these tables there will be no additional columns except legacy ones. All child tables will be called the sections.

3) To add the restrictions to the section tables that define the valid key values for each section. The restrictions do not overlap — no key values apply to several sections at once.

4) To create a key column index for each section. In this study, the indexes were created for the "word" column.

5) To define a trigger to redirect data added to the main table to the corresponding section. Created trigger is work out when SQL command *INSERT* is run.

The created trigger launches a function that adds values to the corresponding section (table). Fragment of the function:

```
CREATE OR REPLACE FUNCTION words_with_reversed_endings_insert_function()
RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.ending = 'ыш') THEN
        INSERT INTO words_with_endings_nii (id, word_form, ending, accent, word_type)
        VALUES (NEW.id, NEW.word_form, reverse(NEW.ending), NEW.accent, NEW.word_type);
    ELSIF (NEW.ending = 'ишй') THEN
        INSERT INTO words_with_endings_niy (id, word_form, ending, accent, word_type)
        VALUES (NEW.id, NEW.word_form, reverse(NEW.ending), NEW.accent, NEW.word_type);
```

The creation of tables, indexes, trigger and function is performed through a *Python* script in an automated mode. The manual adjustment of table and index names is required, since transliterated ending names were used for naming — some cases required the manual intervention (transliterate[9] is used). These cases include, for example, the coincidence of names during transliteration of the endings "*ЕМСЯ*" and "*ЁМСЯ*".

In the context of a *PostgreSQL* database, a trigram is a group of three consecutive characters. We can measure the similarity of the two lines by counting the number of matching trigrams. This simple idea turns out to be very effective for measuring the similarity of words in many natural languages, as well as for solving related problems, such as, for example, *fuzzy search* (search by similarity).

PostgreSQL supports two types of text indexes[10]: *GIN* (*Generalized Inverted Index*) and *GIST* (*Generalized Search Tree*), which provide a work with symbol trigrams, what is prerequisite for using *GIN*, which operates the lexemes by defaults. Despite the fact that the *GIN* by description is very similar to the experiment with inverted strings, *GIST* also has a basis for application: its tree-like structure increases the completeness of search results by including inaccurate hits, which is quite suitable for the rhyme search task, since the table from the work of V.M. Zhirmunsky [22] contains, inter alia, the pairs of endings that do not coincide in spelling.

As part of the search module for a comprehensive analysis of poetic texts, it is possible to add text indexes to solve the following problems:

- search by accentuation mask;
- search by words from the name and text.

The corpus of Pushkin's works was loaded to the database; the main tables with the texts of works and their metadata contain data with a volume of more than 700 rows. The search query includes not only the direct solution of the above problems, but is also adapted for the user to understand: the response array includes additional entries, such as the author's full name and title of the poem, i.e. the request is composite. During the experiment, the query runtime of processing additional parts of the request are not taken into account.

B. Experiments

It is supposed to conduct the following experiments with the search for rhymes in corpuses of the *PostgreSQL* database:

1) To search for the desired ending among the section names: *SELECT * from pg_catalog.pg_tables where %section name conditions%*. It is worth noting that only in this experiment the previously inverted lines described above are used.

2) To search the endings by the incomplete match of *LIKE* on a table without indexes.

3) To search the endings by the incomplete match of *LIKE* on the table with the constructed *GIN* index by symbol trigrams: *CREATE INDEX trgm_idx ON test_trgm USING GIN (t gin_trgm_ops);*

4) To search the endings by the incomplete match of *LIKE* on the table with the constructed *GIST* index by symbol trigrams: *CREATE INDEX trgm_idx ON test_trgm USING GIST (t gist_trgm_ops)*.

For conducting the experiment, the smallest possible sample of 100 examples of endings was taken; 80% of the sample consisted of randomly selected the most frequently used endings (the first 500 one), the remaining 20% were examples from the following 100 used endings (also randomly selected). The time spent on experiments were measured for each of the options (1)–(4). During each experiment, the characteristics are received (the abbreviations are indicated in brackets):

[9] Transliterate – PyPi. https://pypi.org/project/transliterate/

[10] Postgres Pro Standard. https://postgrespro.ru/docs/postgrespro/9.5/textsearch-indexes

- average SQL query runtime (*avg*);
- 50th percentile (*median*);
- 90th percentile (*90 perc*);
- 95th percentile (*95 perc*);
- 98th percentile (*98 perc*).

The results are shown in table I.

TABLE I. THE RESULTS OF THE EXPERIMENT

	Avg	***Median***	***90 perc***	***95 perc***	***98 perc***
Search among section names	0.798	0.831	1.704	1.942	2.381
Without indexes	2.497	2.364	3.450	4.109	4.193
GIN	2.258	2.130	2.974	3.343	3.645
GIST	2.407	2.392	3.179	3.330	3.607

It is possible to make a number of conclusions:

- The least time-consuming option turned out to be (1), suggesting a search among section names. This indicator is partly conditioned by those endings for which the sections were not created — in such cases, the cost of executing the SQL query was negligible.
- Search results without indexes and searches using the *GIST* index differ slightly from each other, what indicates the inappropriateness of using the *GIST* index to solve the research problem.
- Satisfactory results showed the usage of the *GIN* index to search for incomplete matches (3).

An additional experiment on measuring the time which is spent for searching by the accentuation mask or by words from the poems consists in the formation of search queries and comparison their effectiveness. Trigram symbolic indexes *GIST* and *GIN* affected in the query are added separately to the text fields of the tables, namely the fields “Mask of accentuation” and “Text of the poem”; at the first stage of the experiment, the query runtime without indexes was measured. Types of executed requests:

- without indexes;
- using *GIST* index (the operator class *gist_trgm_ops* was used);
- using *GIN* index (the operator class *gin_trgm_ops* was used).

A fragment of a typical SQL query for which runtime was measured:

```
SELECT
    a."ID" as AUTHOR_ID,
    a."LASTNAME", a."FIRSTNAME",
    a."MIDDLENAME",
    p."ID" as POEM_ID, p."NAME" as POEM_NAME,
    m."ACCENTUATION_MASK" FROM "AUTHOR" a
LEFT JOIN
    "POEM" p ON p."AUTHOR_ID" = a."ID"
LEFT JOIN
    "MRSTATISTICS" m ON m."POEM_ID" = p."ID"
WHERE
    m."ACCENTUATION_MASK" LIKE 'cC cC cC c'
```

The query runtime was measured with different variants of the search conditions (for example, a search for a different number of words); the result was an average score of 10 queries with *GIST* and *GIN* indexes. The results of the experiment are shown in table II.

TABLE II. THE RESULTS OF AN EXPERIMENT TO ADD INDEXES TO A SEARCH MODULE

Index name	***Avg query runtime, msec***		
	without indexes	***GIST***	***GIN***
Search by accentuation mask	99	117	116
Search by words from the name and text	129	146	144

The results of an additional experiment showed an increase in the time for processing queries for text values used in the SQL query. Such an increase can be caused either by insufficient test data, or by the inefficiency of the applied indexes within the framework of the problem being solved.

IV. CONCLUSIONS

The usage of *PostgreSQL* built-in database tools has long been limited by search engines in their modern understanding, the results were returned on request in a natural language using a *DBMS* (*Database Management System*). For the task of rhyme search, the program performance is not a determining factor. In the present work, the most prospective approaches were shown, as well as the examples on how to significantly speed up the algorithm using simple steps, what allows other researchers to apply these approaches as part of their research without requiring expert knowledge of the *PostgreSQL* database. In addition, the interface to access the DBMS does not change (except for the manual construction of a table with inverted rows), what is convenient for developers who integrate the text analysis systems with the *PostgreSQL* database.

ACKNOWLEDGMENT

The study was carried with the support of the Russian Science Foundation (project No. 19-18-00466).

REFERENCES

[1] V. Barakhnin and O. Kozhemyakina, “About the automation of the complex analysis of russian poetic text,” CEUR Workshop Proceedings, vol. 934, pp. 167-171, 2012.

[2] N.V. Lapshina, I.K. Romanovich and V.I. Yarkho, “Metrical Handbook for Pushkin’s poems,” M., L.: Academia, 1934.

[3] A. Mittmann, “Escansão automático de versos em português,” Universidade Federal de Santa Catarina, 2016.

[4] D. Fusi, “An Expert System for the Classical Languages: Metrical Analysis Components,” Lexis, vol. 27, pp. 25-45, 2008.

[5] D. Fusi, “A Multilanguage, Modular Framework for Metrical Analysis: IT Patterns and Theorical Issues,” Langages, vol. 3, no. 199, pp. 41-66, 2015.

[6] B. Navarro, “A Computational Linguistic Approach to Spanish Golden Age Sonnets: Metrical and Semantic Aspects,” Proceedings of the Fourth Workshop on Computational Linguistics for Literature, pp. 105-113, 2015.

[7] B. Navarro, M.R. Lafoz and N. Sánchez, “Metrical Annotation of a Large Corpus of Spanish Sonnets: Representation, Scansion and

Evaluation," Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC), pp. 4360-4364, 2016.
[8] M. Agirrezabal, "ZeuScansion: a Tool for Scansion of English Poetry," Journal of Language Modelling, vol. 4, no. 1, pp. 3-28, 2016.
[9] R. Ibrahim and P. Plecháč, "Towards the Automatic Analysis of Czech Verse," Formal Methods in Poetics, Lüdenscheid: RAM-Verlag, pp. 295-305, 2011.
[10] P. Plecháč, "Czech Verse Processing System KVĚTA — Phonetic and Metrical Components," Glottotheory, vol. 7, no. 2, 2016.
[11] I. Pilshchikov, and A. Starostin, "The problems of automation of basic procedures rhythmic parsing accentual-syllabic texts," Russian National Corpus: 2006-2008: New results and perspective, pp. 298-315, 2009.
[12] A. Kurt and M. Kara, "An algorithm for the detection and analysis of arud meter in Diwan poetry," Turkish journal of electrical engineering & computer sciences, vol. 20, no. 6, pp. 948-963, 2012.
[13] M.A. Alnagdawi, H. Rashideh and A.F. Aburumman, "Finding Arabic Poem Meter using Context Free Grammar," Journal of Communications and Computer Engineering, vol. 3, no. 1, pp. 52-59, 2013.
[14] A. Almuhareb, "Recognition of Classical Arabic Poems," Proceedings of the Workshop on Computational Linguistics for Literature, pp. 9-16, 2013.
[15] V.N. Boikov, M.S. Karyaeva, V.A. Sokolov and I.A. Pilshchikov, "On an Automatic Procedure for the Specification of a Poetic Text for an Open Information-Analytical System," CEUR Workshop Proceedings, vol. 1536, pp. 144-151, 2015.
[16] I. Pilshchikov, and A. Starostin, "Reconnaissance automatique des mètres des vers russes: Une approche statistique sur corpus," Langages, vol. 3, no. 199, pp. 89-106, 2015.
[17] K. Taranovsky, "About the relationship between poetic rhythm and topic," About poetry and poetics, Moscow: Languages of Russian culture, pp. 372-403, 2000.
[18] N.I. Golov and L. Ronnback, "SQL query optimization for highly normalized Big Data," Business Informatics, vol. 33, no. 3, pp. 7-14, 2015.
[19] L.I. Lebedev, Yu.V. Yasakov, T.Sh. Utesheva, V.P. Gromov, A.V. Borusjak and V.E. Turlapov, "Complex analysis and monitoring of the environment based on Earth sensing data," Computer Optics, vol. 43, no. 2, pp. 282-295, 2019. DOI: 10.18287/2412-6179-2019-43-2-282-295.
[20] V.B. Barakhnin, O.Y. Kozhemyakina and A.V. Zabaykin, "The Algorithms of Complex Analysis of Russian Poetic Texts for the Purpose of Automation of the Process of Creation of Metric Reference Books and Concordances," CEUR Workshop Proceedings, vol. 1536, pp. 138-143, 2015.
[21] V.B. Barakhnin, O.Yu. Kozhemyakina and I.V. Kuznetsova, "Development and Implementation of the Algorithm for Automatic Analysis of Metrorhythmic Characteristics of Russian Poetic Texts," CEUR Workshop Proceedings, vol. 2523, pp. 290-298, 2019.
[22] V.M. Zhirmunsky, "Rhyme, its history and theory," Petrograd: Academia, 1923.
[23] A.A. Zaliznyak, "Grammatical dictionary of the Russian language. The changing word forms: about 10,000 words," M.: Russian language, 1980.