IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 August 2022, accepted 31 August 2022, date of publication 5 September 2022, date of current version 21 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204388

RESEARCH ARTICLE

Design and Development of Computational Tools for Analyzing Elements of Hindi Poetry

KOMAL NAAZ AND NIRAJ KUMAR SINGH
Department of Computer Science and Engineering, Birla Institute of Technology Mesra, Mesra, Jharkhand 835215, India
Corresponding author: Komal Naaz (komalnaaz1209@gmail.com)

ABSTRACT Poetry writing is a qualitative subject and so is its analysis. Mapping of these poetic elements onto a scale of real numbers is a lacking necessity. Albeit, Hindi literary heritage, being so huge and glorified, there is remarkably very few computational works done exploring the underlying structures. Out of which most of them is to detect a particular metre rather than a generalized approach. The state-of-art metadata generator fails to provide any measures of underlying structural elements of poetry. There is no automated system that generates rhyming pattern hidden in a poem for Hindi language or a system to detect and estimate the extent of figure of speech in a given text of any language. In this article, to extract and evaluate elements of poetry, three efficient tools, namely Text2Mātrā, RPaGen and FoSCal, have been designed and developed. The Text2Mātrā tool provides the numeral scansion for any Hindi input text, which can serve as basis for copious analytical and detection work. RPaGen detects the poem type of any input poem and outputs its rhyming pattern. FoSCal gives a quantitative representation of detected figures of speech in any input text, using the scoring scheme formulated using fuzzy approach and weighted analysis. These tools may find their utility in various fields such as education, literary criticism, philology, authorship-attribution, etc. There have been various computational activities done in the field of poetry analysis over the various languages across the world. However, quantifying the extent of Figure of Speech in poetic compositions, in any language, is entirely a novel approach. Mapping the aesthetic properties of a subjective idea (like poetry) onto a numeral scale, to the best of our knowledge, is first of its kind for Hindi language.

INDEX TERMS Figure of speech, hindi poetry, metre, rhyme, style, tool, weighted analysis, fuzzy, alliteration, pattern.

I. INTRODUCTION

Mammata, an eleventh-century rhetorician defines poetry as flawless meaningful compositions possessing certain desirable qualities and adorned with aesthetic beauties so far as possible. Poet Vishwanath defines poetry as sentence(s) with aesthetic beauty. Later, a seventeenth-century savant Jagannath defines poetry as words propounding amusable meanings [7]. Although these and many other modern scholars differ in their idea of defining poetry, they roughly agree on certain elements like *chanda* or metre, *tuka* or rhyme, and *alaṅkāra* or figure of speech desirable in any poetry [7]. The poetic excellence in such literary pieces is highly attributed to these elements. This research article is a very first attempt towards mapping these agreed-upon considerable ingredients of Hindi poetry onto a numerical scale to bring out its latent qualities in the form of equivalent numeral value(s).

The regular structures contained in the intricacies of mathematics have been a matter of curiosity. In this respect, the disguised structures in poems are no different from mathematical structures [13], [25]. Comparative study of ideas in various forms is the basic instinct of human beings. The purpose of any such study, along with a few others, is to analyse the virtues and vices inherent in these ideas. Expression of feelings by means of poetry is probably the utmost form of communication of thoughts. In this sense, the mathematical and statistical study of the aesthetic aspects contained in literary works, especially poems, is an important justifiable

The associate editor coordinating the review of this manuscript and approving it for publication was Dongxiao Yu.

subject. The analysis of literary works is a common process. These analyses are broadly of two types - literary analysis and computational analysis. Literary analysis is subjective in nature, and is mainly practiced by those who have a direct association with literature. Whereas, computational analysis is objective in nature. The people involved may be related to literature, but they also have additional skills, through which they are also able to study and analyze the various indirect structures embedded in the poems.

This article focuses on the automation of various elements of Hindi poetry namely, *metre* or *chanda*, *rhyme* or *tuka*, and *figure of speech* or *alaṅkāra*. A well-ordered and predetermined miscellany of morae or syllables is referred to as *chanda* or metre. *Mātrika* and *varṅika* are the two most common forms of metres. The *chanda*, which is determined by the ordering and positioning of morae and syllables, is one of the emphases of this article, which can be used as one of the ways to get a numerical equivalent of a given text. For a seamless analysis of the mathematical structures underlying the words of the poetry, an equivalent numerical conversion of the words is required. Any Hindi poem's original text form can alternatively be understood as a series of numerical values *1* and *2*. Such a metamorphosis is governed by a collection of well-defined rules spread throughout numerous literatures concerning the laws of Hindi poetry, rather than being a one-to-one function [7], [11], [19]. Moving on to the next element, rhyme appears to be an undeclared but important feature of contemporary poetry. Rhyming in a poem is determined by a number of aspects, including the position of the rhyming lines in the poem, as well as the words inside the lines. These aspects made the automation process of rhyme pattern generation a non-trivial task. *Alaṅkāra*, or *figure of speech* (FoS), is a Hindi ornamentation that is utilised to improve the elegance and influence of any poetry text. It adds ingenuity to any phrase by making it obscure. The use of rhetorical components enhances aesthetic beauty in poetry. Ornate compositions especially impress the listeners. Its correct use is also considered a sign of intelligence. FoS in Hindi is huge and only a part (alliteration) of it is embraced in this article. These elements are explained in detail in the following sections.

Hindi is the third most spoken language in the world, yet its literary compositions are very less explored computationally [9], [33]. There are some computational works done in modern Hindi [21], [24] but very few in ancient Hindi poetry [5], [12], [15], [16], [17]. Kushwah and Joshi [12] proposed an algorithm for automatic detection of *rola chanda*, but their detection approach is confined to instants count, ignoring the other important elements of poetry (mood, rhyme, figure of speech). In a further work, Joshi and Kushwah [5] proposed the automatic detection algorithm for *caupāī chanda*. Audichya and Saini [15] discusses a technique for automatic metadata generation for Hindi poetry. Impressive work is done by Audichya and Saini [17] in the taxonomic listing of Hindi figures of speech, but insouciant work done in automating it, and in a very similar fashion [16] explained the *chanda* rules in Hindi poetic composition. The aspects of poetry stated above are not exhaustive. It varies greatly from language to language, which implies that algorithms designed for poetry analysis in one language more often may not be equally applicable to another. In contrast to Hindi, a significant amount of work has been done in computing poetic elements of other languages, such as metre detection and classification of Arabic [1], [4], [18] and Persian poetry [28], a study of metre as a stylistic feature in Latin poetry [6], an expert system for harmony test of Arabic poetry [3], a statistical evaluation of Chinese Tang [2], [32] and English [10] poetry, an emotion based classification for Marathi [26], Punjabi [8], and Arabic [20] poetry, a study of rhythm of Tibetan poetry [14]. One of the surprises of this study is that none of the above languages has a computational system or tool that recognises and quantifies figure of speech, a very important component of poetry, which is one of the novelties of this article.

From the literature reviwed, it is to note that, we do not yet have any automated tool to detect the nature of rhymes contained in Hindi poetry. It is clear that numerical transformation of text is desirable when analysis comes into picture. One such text to numerical converter is a part of metadata generator given by Audichya and Saini [15], but an insouciant work done in automating it and lacking to provide an algorithm or a caliber result set. Acting upon the identified gap we created tool that can generate rhyme pattern(s) hidden in an input poem and a tool which outputs a sequence of *1*s and *2*s tantamount to the input text. The use of rhetorical components enhances aesthetic beauty in poetry. Ornate compositions especially impress the listeners. Its correct use is also considered a sign of intelligence. Comparison of the aesthetic beauty generated by ornamentation in two or more compositions is absolutely expected. Such comparisons do happen in literary analysis, but these comparisons are subjective. Therefore, there is room for ambiguity in these analyses. However, this skeptical situation can be avoided if the method of analysis is made objective rather than subjective. There is no tool or any automated system to the best of authors knowledge that estimates the measure of aesthetic components of any input Hindi text that can provide solution to the aforestated problem.

In this article we propose a toolset that is probably the most rigorous computational research done to the present in the area of interest. Three efficient tools are offered, namely Text2Mātrā, RPaGen and FoSCal. The Text2Mātrā tool provides the numeral scansion for any input text, which can serve as the basis for copious analytical and detection (for example, to detect *chanda* type, to detect rhythm pattern, verifying metrical correctness of a given verse, etc.) work. In the current context, RPaGen is the first of its kind, it detects the poem type of any input poem and outputs its rhyming pattern. FoSCal is a tool in a class by itself that estimates the extent of aesthetic components (alliteration) of any input text. The proposed toolset (Text2Mātrā, RPaGen and FoSCal) covers the gaps identified.

The major contributions of the proposed work are:

- A Hindi poetry can be computationally analysed for the various agreed upon elements of poetry. With respect to the first two elements *chanda* and rhyme we created a detection and pattern generating tools (Text2Mātrā and RPaGen) that can help do the automated computational analysis of any poetic piece.
- Using weighted analysis and fuzzy technique we formulated a scoring scheme for the next element, that is, figure of speech (alliteration). This scoring scheme serves as key to the creation of tool (FoSCal) that gives the quantitative estimation of the extent of aesthetic components in any given input text.

The proposed toolset may help to spotlight the rich Indian literary heritage and can act as an attestation of the vast, highly-structured literary history of the nation. Philologists may find its application in pursuing the study of literary texts in order to establish their authenticity and their original form, and determination of their meaning. Poems are an ocean of emotions expressed in very few words that leave anyone affected. Learning poems will help to grow intellect and creativity. Poetry is a healthy way to let out the surging emotions in a growing teen. It helps any learning child to understand the impact of words. Owing to it, this work may find its utility in the field of education and even intended poets or lyricists may find these tools equally helpful. Apart from the educational aspect, these tools are highly serviceable to the community of literary critic. It not only helps them to strengthen their observations but also helps them to create one.

This article has been organized in the following order. Section 2 provides an introduction to Hindi-alphabet and elements of Hindi-Poetry. Algorithms along with their methodology followed by their utility are discussed in section 3. Section 4 talks over implementation and detailed result discussion. Finally, conclusions and future work are put in section 5.

II. A BRIEF INTRODUCTION TO THE HINDI ALPHABET AND ELEMENTS OF HINDI POETRY

A. INTRODUCTION TO HINDI ALPHABET

Hindi-Alphabet is defined as a well-organized set of *akṣara* (letter). An *akṣara* is a root sound that cannot be broken anymore and can be pronounced by one effort of voice. *Akṣara* can be of two types, *svara* (vowels) and *vyañjana* (consonants). There is a total of eleven *svara* and thirty-three basic *vyañjana* in the Hindi-Alphabet (see Table 1) [27], [31].

Svara are *akṣara* that can be pronounced independently whereas *vyañjana* can be pronounced only with the help of *svara*. *Svara* can be further categorized as *hrsva* (short vowel) and *dīrgha* (long vowel) depending on the time required to pronounce them. *Dīrgha svara* requires twice the time needed to pronounce a *hrsva svara* [29].

TABLE 1. Hindi-Alphabet.

Hindi-Alphabet Vowels									
Hrsva svara					*Dīrgha svara*				
अ	इ	उ	ऋ		आ	ई	ऊ	ए	ऐ
[*a*]	[*i*]	[*u*]	[*ṛ*]		[*ā*]	[*ī*]	[*ū*]	[*e*]	[*ai*]
					ओ	औ			
					[*o*]	[*au*]			

Basic Hindi Consonants						
Class	**Sub Class**	**Consonants**				
Sparśa (Plosive)	कवर्ग	क [*ka*]	ख [*kha*]	ग [*ga*]	घ [*gha*]	ङ [*ṅa*]
	चवर्ग	च [*ca*]	छ [*cha*]	ज [*ja*]	झ [*jha*]	ञ [*ña*]
	टवर्ग	ट [*ṭa*]	ठ [*ṭha*]	ड [*ḍa*]	ढ [*ḍha*]	ण [*ṇa*]
	तवर्ग	त [*ta*]	थ [*tha*]	द [*da*]	ध [*dha*]	न [*na*]
	पवर्ग	प [*pa*]	फ [*pha*]	ब [*ba*]	भ [*bha*]	म [*ma*]
Antaḥstha (Sonorant)		य [*ya*]	र [*ra*]	ल [*la*]	व [*va*]	
Uṣma (Sibilant)		श [*śa*]	ष [*ṣa*]	स [*sa*]	ह [*ha*]	

B. BASIC TERMINOLOGIES

There are some basic terminologies that one must know before they dive into *chandas.*

1) SYLLABLE

A consonant along with a vowel or a vowel alone is considered as one syllable. Depending on the vowel (long or short) type a syllable can be long (*guru*) or short (*laghu*).

2) LAGHU OR SHORT SYLLABLE

A short vowel (अ [*a*], इ [*i*], उ [*u*] or ऋ [*ṛ*]) with or without consonant is called a short syllable or *laghu*. To represent *laghu*, in the scansion and metrical analysis, a numerical symbol *1* is used. For example, ह [*ha*], हि [*hi*] etc. are *laghu*.

3) GURU OR LONG SYLLABLE

A long vowel (आ [*ā*], ई [*ī*], ऊ [*ū*], ए [*e*], ऐ [*ai*], ओ [*o*], औ [*au*]) with or without consonant is called a long syllable or *guru*. To represent *guru,* in the scansion and metrical analysis, a numerical symbol *2* is used. For example, ही [*hī*], है [*hai*], etc. belong to *guru*.

4) MĀTRĀ OR MORA OR METRICAL UNIT

It is a unit of metrical quantity. It represents the time required to utter a short vowel. Time required by a short vowel is one mora and time required by a long vowel is two morae [11], [29].

The terms discussed above are required for the understanding of a set of rules discussed in section 3(A), which is used for the creation of *Algorithm 1*.

Algorithm 1 Text_to_mātrā_converter (*line*)

Input: a line of text in Hindi
Output: a corresponding list of numerals 1s and 2s

```
numerals ← empty_list
for i ← 0, length(line) – 1 do
    for j ← 0, length(word[i]) – 1 do
        if word[i][j] in vyanjan or independent_hrsvaSwar then
            numerals · addToList(1)
        end if
        if word[i][j] in laghu then
            continue
        end if
        if word[i][j] in guru or word[i][j] is ',' then
            numerals · removeElement
            numerals · addToList(2)
        end if
        if word[i][j] in independent_dirghaSwar then numerals · addToList(2)
        end if
    end for
end for
return numerals
```

C. ELEMENTS OF POETRY

The emergence of poetry in the human development saga is the result of human being continuously cultivating and becoming deeper and deeper and more organized in the course of communication. Scholars have defined poetry as the verbal means of conveying emotion, which in the context of the human intellect, expresses human feelings in a factual manner [29]. A flat presentation of facts provides ease, but such simplicity is often at the cost of the sensibility and aesthetic beauty of the composition. In this sense, the expression of the feelings of the heart in beautiful, accurate and effective words is called poetry. The elements *mood* or *rasa*, *metre* or *chanda*, *figure of speech* or *alaṅkāra*, and *rhyme* or *tuka* are considered integral parts of poetry (Fig. 1) [7], [23].

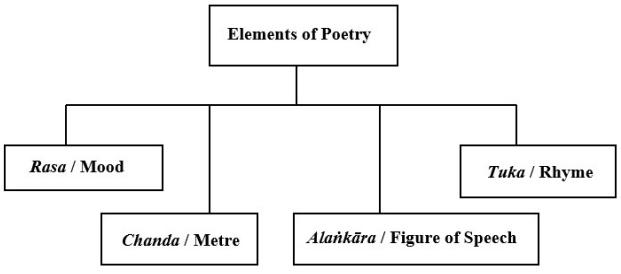

FIGURE 1. Elements of Hindi poetry.

The present article is an attempt towards the design and development of automation tools for analyzing three elements of Hindi poetry. These tools provide a way to identify and extract features (apparent or non-apparent) from among elements that, in some form, are crucial to approximate a subjective idea (such as poetry) into an objective form (such as number(s)). The basics needed to understand the current context is discussed below:

1) CHANDA OR METRE

In Sanskrit and languages (such as Hindi) derived from Sanskrit, *chanda* refers to poetic compositions with a well-ordered and predefined miscellany of morae or syllables. Metres are chiefly of two types, namely, *mātrika* and *varṇika*.

A *mātrika* metre is characterized by having a fixed morae count per metrical line, whereas, a *varṇika* metre is characterized by the count and positions of syllables in each of its metrical lines [11], [29].

2) TUKĀNTATĀ OR RHYMING

It would be surprising to know that at the beginning of such a long history of *chanda* based poetry, there was no accepted practice of rhyme but over a period it became inseparable. Lack of rhyme in singable compositions imply non-elegance and is often perceived as a hindrance to its production.

The presence of rhyme is definitely visible in the compositions of old or ancient Hindi, from where it spontaneously came into the compositions of modern Hindi. At the same time, we also accept that in modern times, many free verse compositions are often free from rhyme. But such compositions are certainly not lyrical.

Details concerning the design of algorithm (*Algorithm 2*) to generate rhyme pattern(s) for any input poem is discussed in section 3(B).

3) ALAṄKĀRA OR FIGURE OF SPEECH

In Hindi, adornment that is used to enhance the elegance of any poetic composition and makes it influential is *alaṅkāra* or *figure of speech* (FoS). It makes any expression abstruse and brings ingenuity to it.

Concepts needed for the understanding of scoring scheme which is used for designing *Algorithm 3* (section 3(C)) are stated below.

Alaṅkāra in Hindi can be broadly classified as *arthālalaṅkāra*, *śabdālaṅkāra* and *ubhayalaṅkāra*. *Śabdalaṅkāra* can be further categorized as character-oriented (*varṇa mūlaka*) and word-oriented (*śabda mūlaka*). The first FoS tool for Hindi language, proposed in this article, covers character-oriented *śabdālaṅkāra* or alliteration, which is further categorized into four types: *chekānuprāsa, vṛttyanuprāsa, śrutyanuprāsa* and *antyānuprāsa* (see Fig. 2).

a: Chekānuprāsa

If a group of consonants is repeated once in both form and sequence then *Chekānuprāsa* is present. For example, बाल बेली सूखी सुखद, इहि रूखे रुख घाम [*bāla belī sūkhī sukhada*, *ihi rūkhe rukha ghāma*]

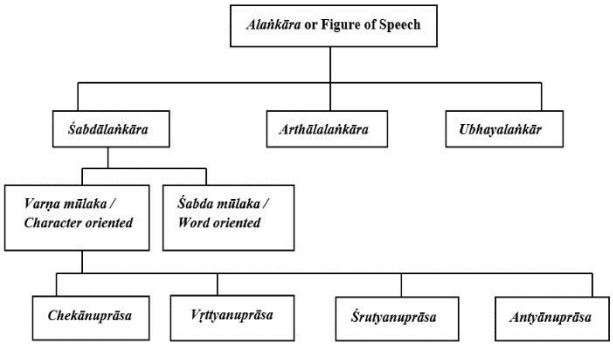

FIGURE 2. ***Alaṅkāra:*** **Types and Subtypes.**

Here ब, ल in बाल बेली, स, ख in सूखी सुखद, and र, ख in रूखे रुख have single repetition where order and form are maintained, so *chekānuprāsa* is present.

b: Vṛttyanuprāsa

If one consonant is repeated once or several times; a group of consonants repeated once or several times in the form; a group of consonants repeated several times in both form and sequence then *Vṛttyanuprāsa* is present.

The above definition can be simplified into five cases:

Case 1: One consonant is repeated once. For example,
उघरहिं बिलाम बिलोचन ही के [*ugharahiṃ b ilāma bilocana hī ke*]
Here, ब is repeated once.

Case 2: One consonant is repeated multiple times. For example,
सबहि सुलभ सब दिन सब देसा [*sabahi sulabha saba dina saba desā*]
Here, स is repeated multiple times.

Case 3: A group of consonants is repeated once in the form. For Example,
रास सरिता कब बक अवगाहहिं [*rāsa saritā kaba baka avagāhahiṃ*]
Here, "रस सर" and 'कब बक' have single repetition but the sequence in group of consonants is not maintained.

Case 4: A group of consonants is repeated multiple times in the form. For example,
उस प्रमद के अलकदाम से मादक सुरभि निकलती [*usa pramada ke* alakadāma *se mādaka surabhi nikalatī*]
Here, 'मद दम मद', is a group of consonants repeated multiple times in the form but without order.

Case 5: A group of consonants is repeated multiple times in the form and sequence.
For example,
स्व काकली को कलकंठ कोकिला [*sva kākalī ko kalakaṇha kokilā*]
Here, "क क ल" is a group of consonants repeated multiple times in the form and with sequence.

If any of the five cases is established in a poem then *Vṛttyanuprāsa* is present [7].

c: Śrutyanuprāsa

If sounds belonging to specific articulation point(s) dominate then *Śrutyanuprāsa* is present. For example,
तुलसीदास सीदत निसि दिन देखत तुम्हारी निठुराई [*tulasīdāsa sīdata nisi dina dekhata tumhārī niṭhurāī*]

In this line, letters belonging to the dental class, in sequence, are त ल स द स स द त न स द न द त त न, which make 16 out of total 23 letters, that is, 69.56 percent of letters belong to the dental class. So, it is fair to say letters from the dental class dominate and hence *śrutyanuprāsa* is present [23].

d: Antyānuprāsa

The presence of rhyme, to any extent, is realized by *antyānuprāsa* in the corresponding piece of literature. For example,
तैं रहीम मन आपुनो, कीन्हों चारु चकोर [*taiṃ rahīma mana āpuno, kīnoṃ cāru cakora*]
निसि बासर लागो रहै, कृष्णचंद्र की ओर [*nisi bāsara lāgo rahai, kṛṣṇacandra kī ora*]

Here, चकोर and ओर form rhyming between the two lines and so we can say *antyānuprāsa* is present.

III. ALGORITHMS AND METHODOLOGY

This section contains the various algorithms designed by referring to the rules specified across a number of sources along with their methodology. This discussion is followed by the utility of the proposed tools.

The asymptotic computational complexities, in terms of space and time, of the tools are discussed just below the algorithm specifications of the respective tools.

A. CONVERTING A HINDI METRICAL LINE INTO EQUIVALENT NUMERICAL SCANSION

Algorithm *Text_to_mātrā_converter* converts a given line of Hindi text into an equivalent sequence of *1*s and *2*s. It assigns *1*s and *2*s for all *laghus* and *gurus* respectively except for a few special cases where a *laghu* is assigned a weight of *2*. These special cases are [11], [29]-

1. If a vowel is followed by an *anusvāra*. For example, in the word कंक [*kaṅka*] the syllable कं [*kaṅ*] has *mātrā* value *2*.
2. If a vowel is followed by a *visarga*. For example, तः [*taḥ*] in the word अतः [*ataḥ*] has *mātrā* value *2*.
3. If a vowel is followed by a conjunct consonant. For example, ब [*ba*] in the word बन्ध [*bandha*] is to be considered long, since it is followed by the conjunct न्ध [*ndha*].

Algorithm 1 converts the given input *line* into corresponding list of numerals, *numerals*. Here *vyanjan* is a list of all consonant sounds of Hindi Alphabet, *laghu* is a list of all short vowels, *guru* is a list of all long vowels, *independent_hrsvaSwar* is a list of all *hrsav svara* and *independent_dirghaSwar* is a list of all *dīrgha svara. addToList()* is a function that inserts the given item into the list. *removeElement()* is a function that pops out element from the list.

Referring to Algorithm 1, for a given input 'दुख में सुमिरन सब करे' [*dukha meṃ sumirana saba kare*], the corresponding output will be [1 1 2 1 1 1 1 1 1 1 2].

The algorithm *Text_to_mātrā_converter* (*line*) takes a line of text as input. Let *m* and *k* denote the length of line (in terms of word count) and the maximum length of a word in that line respectively. Line number 1 is a declaration statement. Line number 2 is the beginning of a for loop, that executes *m* times. Line numer 3 is an inner for loop having frequency count *k*. All the lines in the algorithm from line 4 to 11 are statements that requires constant time. Line 15 is a constant time return statement. Thus the overall time required by the algorithm is majorly determined by the time elapsed in the nested loop, which is $m \cdot k$, resulting in $O(m \cdot k)$ time complexity.

For all practical purposes the variables *m* and *k* may be treated as constants rather as variables.

B. DETERMINING THE RHYMING QUALITY IN HINDI POETRY

In poetry, metrical or non-metrical, the end of their lines should also be according to the rules. Rhymes are defined to be of three types, namely, *best*, *medium*, and *worst* (see Table 2) [29]. The rhyming type is defined as *not-present* if none of the preceding types meet. Depending upon the sequencing of lines contributing to rhyming, a poem can be of multiple types. The proposed algorithm detects the rhyme pattern in a given input poem. The heterogeneity in poem structure makes this algorithm a non-trivial one.

Rhyme is typically checked on the last words of concerned poetic lines. If the last words happen to be the same, we proceed with the second last words and so on until we get a non-identical word pair. We have categorized the rhyme pattern on the line level into three classes, namely, *RP1* (Rhyme Pattern 1), *RP2* and *RP3*. If the last word of both lines is different, they fall in class *RP1*. In case, the last words are identical but the second last words are not, then they fall in class *RP2*, else they fall in class *RP3*. To supplement this discussion, in Table 3, rhyme-making words (or phrases) are underlined for clarity.

As mentioned earlier in this section, based upon length of each stanza and/ or position of lines participating in making rhymes, poetry can be of numerous types. In this section, we have proposed an algorithm *Rhyme_pattern_generator* to recognize the three most common poem types named *PT1* (Poem Type 1), *PT2* and *PT3*. Rest of the poem types will fall under *unknown* type.

TABLE 2. Example of rhyming words (different categories).

Rhyming			
Best	**Medium**	**Worst**	**Not Present**
कैसे [*kaise*], जैसे [*jaise*]	NA	कैसे [*kaise*], किसे [*kise*]	कैसे [*kaise*], काई [*kāī*]
काई [*kāī*], सचाई [*sacāī*]	NA	कैसे [*kaise*], हमसे [*hamase*]	आस [*āsa*], जागत [*jāgata*]
आवत [*āvata*], जावत [*jāvata*]	सूचना [*sūcanā*], बूझना [*būjhanā*]	देखिये [*dekhiye*], सोचये [*socaye*]	साजन [*sājana*], जागत [*jāgata*]
बरसत [*barasata*], तरसत [*tarasata*]	विहंसत [*vihaṃsata*], हुलसत [*hulasata*]	अरुचित [*arucita*], तड़पत [*tadapata*]	हुलसत [*hulasata*], आगमन [*āgamana*]

TABLE 3. Examples of different rhyme patterns.

Class	Line 1	Line 2
RP1	अधरों की आतुरता में ही जब आभासित हो प्याला [*adharoṃ kī āturatā meṃ hī jaba ābhāsita ho pyālā*]	रहे न हाला, प्याला, साकी, तुझे मिलेगी मधुशाला [*rahe na hālā, pyālā, sākī, tujhe milegī madhuśālā*]
RP2	घर-घर देखा धुआं पर, सुना, विश्व में आग लगी है [*ghara-ghara dekhā dhuāṃ para, sunā, viśva meṃ āga lagī hai*]	'जल ही जल' जन-जन रटता है, कंठ-कंठ में प्यास जगी है ['*jala hī jala' jana-jana raṭatā hai, kaṇṭha-kaṇṭha meṃ pyāsa jagī hai*]
RP3	मैं बेकरार हूं आवाज में असर के लिए [*maiṃ bekarāra hūṃ āvāja meṃ asara ke lie*]	ये एहतियात जरूरी है इस बहर के लिए [*ye ehatiyāta jarūrī hai isa bahara ke lie*]
RP3	देख, दहलीज से काई नहीं जाने वाली [*dekha, dahalīja se kāī nahīṃ jāne vālī*]	ये खतरनाक सचाई नहीं जाने वाली [*ye khataranāka sacāī nahīṃ jāne vālī*]

- *Poem Type 1*: Poems in which rhyme is expected within each consecutive pair of lines (couplets) come under this type. If there are *n* lines in a poem then rhyme is expected between line *1* & line *2*, line *3* & line *4*, line *5* & line *6*, …, and line *n*-1 & line *n*. *Dohā*, *Caupāī*, *Bhujaṃgaprayāta* are some of the common metre-based poem types that fall under this class. Here, the number of lines in the poem should be a multiple of two.
- *Poem Type 2*: Poems in which rhyme is expected between even lines of each quadruplet come under this type. If there are *n* lines in a poem then rhyme is expected

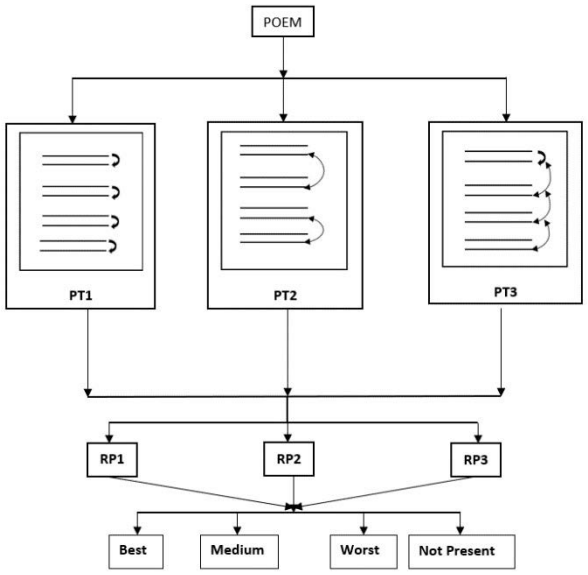

FIGURE 3. Overview of rhyme pattern extraction.

between line *2* & line *4*, line *6* & line *8*, ..., and line *n-2* & line *n*. Here, the number of lines in the poem must be a multiple of four.

- *Poem Type 3*: In this type, given a poem of length *n*, rhyme is expected between lines *1* & 2, followed by lines *2* & 4, lines *2* & 6 up to lines *2* & *n*. Here, the number of lines must be more than four and a multiple of two.

Altogether, given a poem as input, first its type (*PT1*, *PT2*, *PT3* or *unknown*) gets checked. Once we get the poem type, we check the rhyme pattern (*RP1*, *RP2* or *RP3*) on the desired lines. Based on the rhyme pattern, it yields the words on which rhyme will be checked and classified as *best*, *medium*, *worst* or *not present* (Fig. 3).

Algorithm 2 takes poem as input and returns rhyme pattern(s) of the poem. It checks the poem type and invokes functions (*checkPT1(lines), checkPT2(lines) and checkPT3(lines)*) to extract rhyme out of it accordingly. Algorithm for the invoked functions is presented as *Algorithm 2.2, 2.3* and *2.4*. Here, *getRhyme(line1, line2)* is a function that returns the rhyme class. *randomCheck(poem)* is a function that checks the poem type on the randomly picked lines of the given poem.

To simplify, *Algorithm 2* checks poem type and accordingly calls one among *Algorithm 2.2, 2.3 and 2.4,* which in turn calls *Algorithm 2.1*.

Algorithm 2.1 checks the rhyme pattern on the line level and the rhyme class for the input lines (*line1, line2*). Here, *getRP()* is a function that returns the rhyme pattern type, *getLastWord()* is a function that returns the word from the end of line on which rhyme will be checked and *syllabifier()* is a function that returns the list of syllables of a word.

Algorithm 2.2 returns the rhyming pattern for the input poem (*lines*) according to *PT1*. It calls *checkRhyme(), Algorithm 2. 1*, to get the rhyme pattern on the line level.

Algorithm 2 Rhyme_pattern_generator (*poem*)

Input: Poem
Output: Rhyme pattern(s) of the input poem

lines ← *list_of_lines*(*poem*)
n ← *total_lines*(*poem*)
if *n* % 2 == 0 **then** *x* ← *checkRhyme*(*lines*[0], *lines*[1])
  y ← *checkRhyme*(*lines*[2], *lines*[3])
  z ← *checkRhyme*(*lines*[1], *lines*[3])
  p ← *checkRhyme*(*lines*[0], *lines*[3])
  if *x* **then**
    if *y* = 0 and *p* **then**
      decision ← 3
    else if *y* **then** *decision* ← 1
    else *decision* ← *randomCheck*(*lines*)
    end if
  else if *z* and *n* %4==0 **then** *decision* ← 2
  else *decision* ← *randomCheck*(*lines*)
  end if
  if*decision* == 1 **then** *checkPT*1(*lines*)
  else if *decision* == 2 **then** *checkPT*2(*lines*)
  else if*decision* == 3 **then** *checkPT*3(*lines*)
  else **return** "*UnknownType*"
  end if
return "*InvalidInput*"

Algorithm 2.1 checkRhyme(*line1, line2*)

Input: lines on which rhyming is to check
Output: Rhyming pattern and rhyme class for the input lines

RP ← *getRP* (*line*1, *line*2)
*word*1 ← *syllabifier*(*getLastWord*(*line*1, *RP*))
*word*2 ← *syllabifier*(*getLastWord*(*line*2, *RP*))
for *i* ← 0, *min*(3, *length*(*word*1), *length*(*word*2)) **do**
  *syll*1 ← *word*1[*length*(*word*1) −*i*]
  *syll*2 ← *word*2[*length*(*word*2) −*i*]
  if *syll*1 == *syll*2 **then**
    score ← *score* + 2
  else if *syll*1 and *syll*2 have same *vowel* **then**
    score ← *score* + 1
  end if
end for
if *score* > 4 **then**
  return[*RP*, '*Best*']
else if *score* < 4 and *score* > 2 **then**
  return[*RP*, '*Medium*']
else if *score* < 2 and *score* > 0 **then**
  return[*RP*, '*Worst*']
else
  return[*RP*, '*NotPresent*']

Algorithm 2.3 returns the rhyming pattern for the input poem (*lines*) according to *PT2*.

Algorithm 2.2 checkPT1(*lines*)
Input: set of lines
Output: returns the rhyming sequence according to poem pattern 1
1: **for** $i \leftarrow 0$, *length*(*lines*) $- 1$, by 2 **do**
  $x \leftarrow$ *checkRhyme*(*line*[i], *line*[$i + 1$])
  addToList $\cdot$ *result*(x)
2: **end for**
3: **return** *result*

Algorithm 2.3 checkPT2(*lines*)
Input: set of lines
Output: returns the rhyming sequence according to poem pattern 2
1: **for** $i \leftarrow 0$, *length*(*lines*) $- 1$, by 4 **do**
  $x \leftarrow$ *checkRhyme*(*line*[$i + 1$], *line*[$i + 3$])
  addToList $\cdot$ *result*(x)
2: **end for**
3: **return** *result*

Algorithm 2.4 checkPT3(*lines*)
Input: set of lines
Output: returns the rhyming sequence according to poem pattern 3
1: $x \leftarrow$ *checkRhyme*(*lines*[0], *lines*[1])
2: $y \leftarrow$ *checkRhyme*(*lines*[0], *lines*[3])
3: *addToList* $\cdot$ *result*(x)
4: *addToList* $\cdot$ *result*(y)
5: **for** $i \leftarrow 3$, *length*(*lines*) $- 1$, by 2 **do**
  $x \leftarrow$ *checkRhyme*(*line*[i], *line*[$i + 2$])
  addToList $\cdot$ *result*(x)
6: **end for**
7: **return** *result*

Algorithm 2.4 returns the rhyming pattern for the input poem (*lines*) according to *PT3*.

The procedure of extraction of rhyme pattern for a given input poem is depicted in Fig. 4.

The sub-algorithm *checkRhyme*(*line1, line2*) takes a pair of lines as input. Line 1 is a call to constant time function *getRP(line1, line2)*. Lines 2 and 3 call the O(k) time complexity function *syllabifier(word)*. The for loop in line 4 runs for constant time (maximum 3 iteration). All of the remaining operations require constant time resulting in O(k) overall complexity. The sub-algorithm *checkPT1*(*lines*) receives input consisting of l number of lines. The for loop at line 1 runs for (l/2) + 1 times wherein *checkRhyme(line1, line2)* is invoked. Remaining all operations here have constant computational times resulting in O($l \cdot k$) time complexity for sub-algorithm *checkPT1*(*lines*). Following the similar observations, the sub-algorithms *checkPT2*(*lines*) and *checkPT3*(*lines*) result in O($l \cdot k$) time complexity.

The algotithm *Rhyme_pattern_generator* (*poem*) takes the entire poem as its input. Lines 1 and 2 are constant time assignment operations. The conditional statement at line 3 requires constant time and the assignment operations to variable *x, y, z* and *p* invoke *checkRhyme(line1, line2)* serially, requiring O($k \cdot l$) time by each assignment operation. All computation operations from line 4 to 11 require constant times. Line 12 invokes *checkPT1(lines)*, line 13 invokes *checkPT2(lines)* and line 14 invokes *checkPT3(lines)* and each of them have O($k \cdot l$) time complexity. Remaining operations require constant time, resulting in overall O($k \cdot l$) complexity for *Rhyme_pattern_generator* (*poem*).

C. DETERMINING PRESENCE AND EXTENT OF RHETORICAL ELEMENTS IN HINDI POETRY

Figures of Speech (*alaṅkāra*) make significant contribution to the art of poetics and are recognized as a prominent element of poetry. The presence of rhetorical elements very often gets reflected in the enhanced suaveness of poetic compositions. Measuring the extent of such elegance is an important idea as literary compositions with a significant presence of rhetorical elements tend to enthral their receiver. Mapping this subjective property onto a numeral scale may help in performing comparative analyses of such literary pieces. The proposed tool provides a quantitative measure of the presence and extent of rhetorical elements in any text, giving a score ranging from zero to one, where zero and one correspond to lowest and highest score respectively.

The tool focuses on alliteration, a type of *alaṅkāra* that is based on the repetition of one or more *akṣara*. Alliteration can be of four types (discussed in section 2); the score is calculated for each type independently and then fused to get the final score for the entire poem. Scoring methodology, propounded by authors of this article, for each type is discussed below.

Antyānuprāsa is present in the text if rhyming is present. The algorithm for generating a rhyme pattern for a given poem is discussed in section 3(B). Rhyming can be *best*, *medium*, *worst* or *not present*. It is obvious that the best rhyme in any poem will be more captivating than one with inferior rhyme (*medium*, *worst* or *not present*). So, the *best* rhyme is scored *1*, which is the highest scoring value and *not present* is scored *0*, which is the lowest scoring value. *Medium* and *worst* types are scored $^2/_3$ and $^1/_3$ respectively. This assignment is justified by the fact that these weights are equidistant on the number scale with extreme values zero and one (see Fig. 5).

Sounds in Hindi are classified into six classes based on their place of articulation (see Table 4). *Śrutyanuprāsa* is realized when sounds belonging to specific articulation points dominate. The majority implies dominance, but in some cases, even non-majoritarian may dominate. Based on context, function *checkShrutya(text,d)* is supplied with a suitable value for dominance (i.e., d). Recommended dominance value is forty percent or above.

For a particular d value setting (say forty percent), the presence of *śrutyanuprāsa* may be acknowledged by the

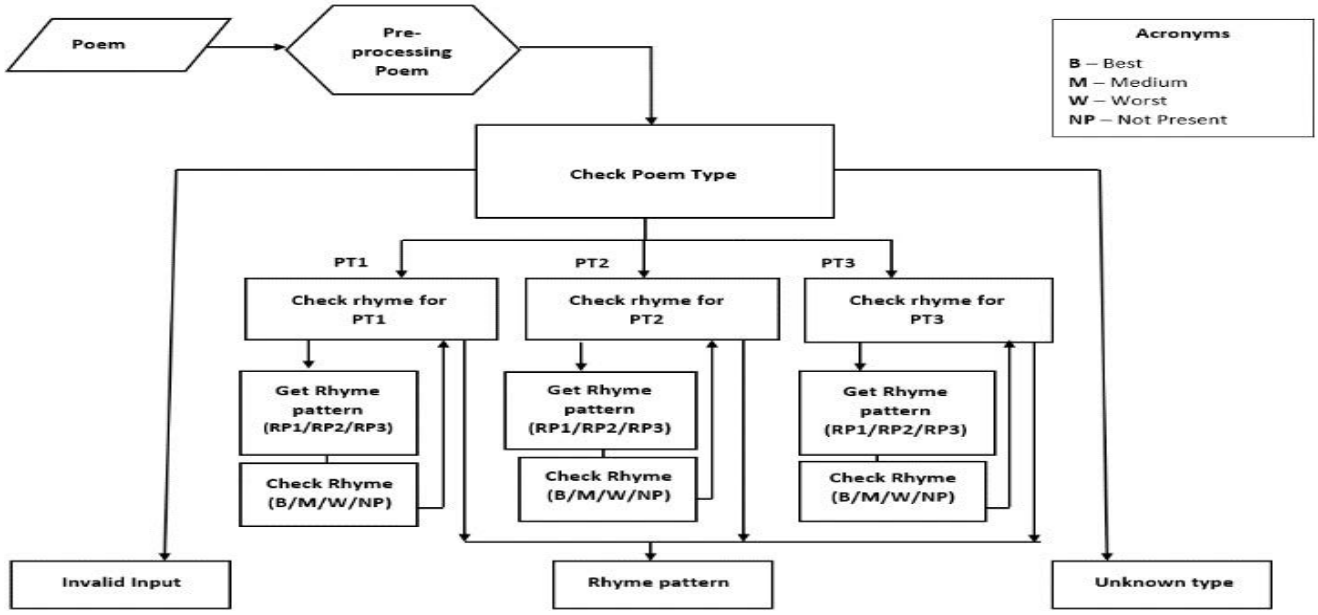

FIGURE 4. Flow diagram for Rhyme_pattern_generator.

TABLE 4. Hindi-alphabet based on place of articulation.

Point of Articulation	Hindi Alphabet											
Velar	अ	आ	क	ख	ग	घ	ङ	ह	ऐ	ए	ओ	औ
	[*a*]	[*ā*]	[*ka*]	[*kha*]	[*ga*]	[*gha*]	[*ṅa*]	[*ha*]	[*ai*]	[*e*]	[*o*]	[*au*]
Palatal	इ	ई	च	छ	ज	झ	ञ	य	श	ऐ	ए	
	[*i*]	[*ī*]	[*ca*]	[*cha*]	[*ja*]	[*jha*]	[*ña*]	[*ya*]	[*śa*]	[*ai*]	[*e*]	
Retroflex	ऋ	ट	ठ	ड	ढ	ण	र	ष				
	[*ṛ*]	[*ṭa*]	[*ṭha*]	[*ḍa*]	[*ḍha*]	[*ṇa*]	[*ra*]	[*ṣa*]				
Dental	त	थ	द	ध	न	ल	स	व				
	[*ta*]	[*tha*]	[*da*]	[*dha*]	[*na*]	[*la*]	[*sa*]	[*va*]				
Bilabial	उ	ऊ	प	फ	ब	भ	म	व	ओ	औ		
	[*u*]	[*ū*]	[*pa*]	[*pha*]	[*ba*]	[*bha*]	[*ma*]	[*va*]	[*o*]	[*au*]		
Nasal	ङ	ञ	ण	न	म							
	[*ṅa*]	[*ña*]	[*ṇa*]	[*na*]	[*ma*]							

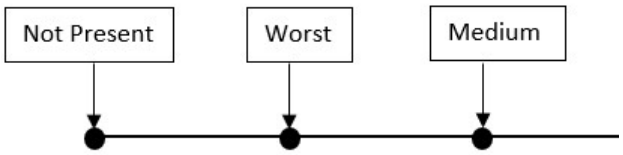

FIGURE 5. Weight assignment for different rhyme types.

score value *1* if calculated dominance exceeds 0.4, and zero otherwise. However, from a practical viewpoint, it is unfair to assign a score of zero for the case where calculated dominance is slightly lower than the required threshold. Hence, scoring for *śrutyanuprāsa* is not crisp but a fuzzy problem and for the same reason, the right degenerate trapezoidal function (Fig. 6) is used to score the presence of *śrutyanuprāsa* in any input text.

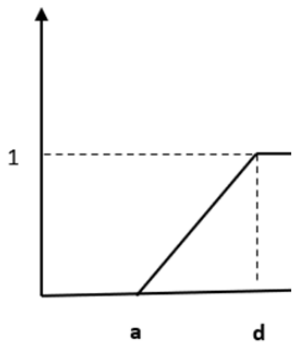

FIGURE 6. Right degenerate trapezoidal function.

Let a and d represent x coordinates of membership function μ. Then,

Right degenerate Trapezoidal $(x; a, d) = 0$ if $x < a$;

$= (x - a)/(d - a)$ if $a \leq x \leq d$;

$= 1$ if $x > d$;

$$\mu(x) = \max(\min(x - a/d - a, 1), 0) \quad (1)$$

The $\mu(x)$ value calculated using (1) yields the desired Śrutyanuprāsa Score (śScore)

Here, d is the point of dominance set by the user and a is the lower bound calculated by assuming the uniform distribution of sounds across all the available classes.

$a = 1$/number of place of articulation classes

$a =$ ceil(1/6) $= 0.17$

Vṛttyanuprāsa is yet another prominent *alaṅkāra* type, and is further classified into five different subtypes (refer section 2); *chekānuprāsa* is one of these five subtypes. Scoring methodology for *vṛttyanuprāsa* and *chekānuprāsa* goes as follows [7]:

Let,

n_1 = no. of words where first or last consonant have multiple repetition

n_2 = no. of words where first or last consonant have single repetition

n_3 = no. of words where group of consonants have multiple repetition

n_4 = no. of words where group of consonants have single repetition

n_5 = no. of words where ordered group of consonants has multiple repetition

n_6 = no. of words where ordered group of consonants have single repetition

tw = no. of words in the input text

Then,

$S_i = n_i/\text{tw}, \quad$ where $1 \leq i \leq 5$

vṛttyanuprāsaScore (vScore) $= \max\{S_i\}$

chekānuprāsaScore(cScore) $= n_6/\text{tw}$

antyānuprāsaScore(aScore) $\in \{0, 1/3, 2/3, 1\}$

Total figure of speech score for the poem is:

alaṅkāraScore (alScore)

$$= [((\text{śScore} + \text{vScore} + \text{cScore})/3) + \text{aScore}]/2 \quad (2)$$

The final *alaṅkāra* score for the input poem is given using (2).

Algorithm 3 gives the *alaṅkāra* score (poemScore) of any input text (*poem*). The algorithm invokes various functions where, *multiOdrdered(line)* is a function that returns two lists, first is the list of words having group of *vyañjana* with muptiple repetition and second is the list of words having group of *vyañjana* with only one repetition. And order of *vyañjana* is maintained in both cases. *unordered(line)* is a function that returns two lists, first is the list of words having a group of

Algorithm 3 Alaṅkāra_score(*poem*)

Input: Poem

Output: alaṅkāra score of the input poem

for $i \leftarrow 0$, *no_of_lines*(*poem*) $- 1$ **do**
  totalWords $\leftarrow$ *total_Words*(*line*[*i*])
  tempResult $\leftarrow$ *multiOrdered*(*line*[*i*])
  if *tempResult*[0] $\neq$ *empty* **then**
    *S*5 $\leftarrow$ *length*(*tempResult*[0])/*totalWords*
  end if
  if *tempResult*[1] $\neq$ *empty* **then**
    chekānuprāsaScore $\leftarrow$ *length*(*tempResult*[1])/*totalWords*
  end if
  tempResult $\leftarrow$ *unordered*(*line*[*i*])
  if *tempResult*[0] $\neq$ *empty* **then**
    *S*4 $\leftarrow$ *length*(*tempResult*[0])/*totalWords*
  end if
  if *tempResult*[1] $\neq$ *empty* **then**
    *S*3 $\leftarrow$ *length*(*tempResult*[1])/*totalWords*
  end if *tempResult* $\leftarrow$ *firstLast*(*line*[*i*])
  if *tempResult*[0] $\neq$ *empty* **then**
    *S*2 $\leftarrow$ *length*(*tempResult*[0])/*totalWords*
  end if
  if *tempResult*[1] $\neq$ *empty* **then**
    *S*1 $\leftarrow$ *length*(*tempResult*[1])/*totalWords*
  end if
  vṛttyanuprāsaScore $\leftarrow$ *max*(*S*1, *S*2, *S*3, *S*4, *S*5)
  tempResult $\leftarrow$ *checkShrutya*(*line*[*i*])
  $a \leftarrow$ *totalVyanjan* $* 0.4$
  $b \leftarrow$ *totalVyanjan* $* 0.17$
  śrutyanuprāsaScore $\leftarrow$ *max*(*min*(((*tempResult* $- a$)/$(b - a)$), 1), 0)
  totalScore $\leftarrow$ (*chekānuprāsaScore* + *vṛttyanuprāsaScore* + *śrutyanuprāsaScore*)/3
  add $\cdot$ *result*(*totalScore*)
end for
tempResult $\leftarrow$ *Rhyme_pattern_generator*(*poem*)
for *i, tempResult*[1] **do**
  if *i* == 'Best' **then**
    score[*i*] $\leftarrow 1$
  else if *i* == 'Medium' **then**
    score[*i*] $\leftarrow 2/3$
  else if *i* == 'Worst' **then**
    score[*i*] $\leftarrow 1/3$
  else
    score[*i*] $\leftarrow 0$
  end if
end for
antyānuprāsa $\leftarrow$ *sum*(*score*)/*length*(*Score*)
poemScore $\leftarrow$ ((*sum*(*result*)/*length*(*poem*)) + *antyānuprāsa*)/2
return *poemScore*

vyañjana with multiple repetitions and second is the list of words having a group of *vyañjana* with only one repetition. *firstLast(line)* is a function that returns two lists. First list contains words that have the first or last *vyañjana* repeated more than once and the second list contains words that have the first or last *vyañjana* repeated only once. *checkShrutya(text,d)* is a function that takes text and dominance value as parameter and returns the highest number of consonants belonging to one of the six points of the articulation class.

The algotithm *Alaṅkāra_score*(*poem*) takes an entire poem as input. The loop at line 1 runs for l (the count of lines in input poem) times. The assignement inside the loop to the variable *totalWords* requires constant time, whereas the assignment to *tempResult* is a call to O(m) time function, *multiOredered(line[i])*. Lines from 2 to 5 are constant time statements. *tempResult* assignment in line 5 is a call to O(m) time function, *unoredered(line[i])*. All the statements from line 6 to 9 require constant time. The assignment operation in line 9 invokes the function *firstLast(line[i])* and its timing requirement is constant. Lines 10 to 14 are statements that require constant time. *Rhyme_pattern_generator (poem)* is invoked at line 15 and as analysed earlier the time complexity of this function evaluates to O($k \cdot l$). Line 16 is the beginning of an another loop that runs for maximum *l/2* times. Lines 17 to 25 are all statements with constant time. Thus, the overall time complexity of the algorithm *Alaṅkāra_score*(*poem*) is maximum{O($l \cdot m$), O($k \cdot l$)}.

The process of *alaṅkāra* score calculation for a given input poem is depicted in Fig. 7.

A discussion on the utility of the proposed tools:

- *Algorithm 1* converts a given text into equivalent numeral scansion consisting of values *1*s and *2*s only; thereby opening scope for quantitative analysis of qualitative data.
- Poems employ aesthetic and rhythmic elements of language, such as phonesthetics, sound symbolism, and metre, to evoke meanings in addition to, or instead of, a mundane apparent meaning. In the process of making the poem, technicality or the rhythm can be missed, the above algorithms can help to detect the flaw, if any, in the poem.
- The numeral scansion produced by *algorithm 1*, rhyming pattern by *algorithm 2* and figure of speech score by *algorithm 3* automate and supplement the comparative study of style among a number of poetic compositions by different authors.
- These tools may serve as an early model for the automated literary critic.

IV. IMPLEMENTATION AND RESULT ANALYSIS

This section contains implementations of algorithms discussed in section 3. The robustness of these implementations is ensured by subjecting them to a rigorous validation process over an all-inclusive set of inputs. The implementations are done using python programming language on Google Colaboratory.

A. TOOL 1: Text2Mātrā

Text2Mātrā tool, an implementation of algorithm *Text_to_mātrā_converter*, generates numeral scansion for a given text in Hindi. As an illustration, the input-output for four representative *chanda* types are given in Table 5 [36].

The correctness of the output produced may be validated against the rules for corresponding *chanda* type.

- *Dohā* is a moraic metre with two metrical lines and each line having two quarters, resulting in a total of four quarters. First quarter of each line or odd quarters have a total of thirteen instant (*mātrā)* counts and the second quarter of each line or even quarters have a total of eleven instant (*mātrā*) counts; thereby resulting in twenty-four instant counts per metrical line. Input 1 from Table 5 is a *dohā chanda* and the output scansion clearly corroborates the aforestated rules.
- *Caupāī* is one of the very famed moraic metre consisting of four quarters. Each quarter totals sixteen instant counts and any quarter must not end with 121 [*laghu, guru, laghu*] or 221 [*guru, guru, laghu*]. Input 2 from Table 5 is a *caupāī* and the output clearly stands for the stated rules.
- *Śloka* is a well-known syllabic metre (profusely used in *vedic literature*) consisting of two lines. A line has two quarters and each quarter is formed of eight syllables. Every odd quarter must have fifth, sixth and seventh syllables as [*laghu, guru, guru*] i.e., 122 and every even quarter must have fifth, sixth and seventh syllables as [*laghu, guru, laghu*] i.e., 121. Input 3 from Table 5 is a *śloka* and its output scansion follows the given rules.
- *Bhujaṃgaprayāta* is a syllabic metre where each line is formed out of four repetitions of [*laghu, guru, guru*] i.e., 122, resulting in twelve syllables for each line. Input 4 from Table 5 is an example of *bhujaṅgaprayāta chanda* and the output scansion validates the pre-existent rules.

B. TOOL 2: RPaGen (RHYME PATTERN GENERATOR)

RPaGen tool, an implementation of algorithm *Rhyme_pattern_generator* extracts rhyme pattern from given Hindi poem. As an illustration, the input-output for a few representative poems is given in Table 6. A sufficiently large number of valid and invalid inputs were provided to check the robustness of the tool and some of them are listed in Table 6 [36].

Poem in example 1 (Table 6) belongs to class *PT3*. Therefore, lines 1 and 2 should rhyme. Further, there are two ways to get the rhyme pattern of the input poem, first is when line 1 rhymes with lines 4, 6 and 8 and the second is when line 2 rhymes with lines 4, 6 and 8. The tool provides both possible rhyming patterns. So, when you look at the output of example input 1 (Table 6), it is a list of three elements. First is the rhyming pattern and rhyme class between lines 1 and 2,

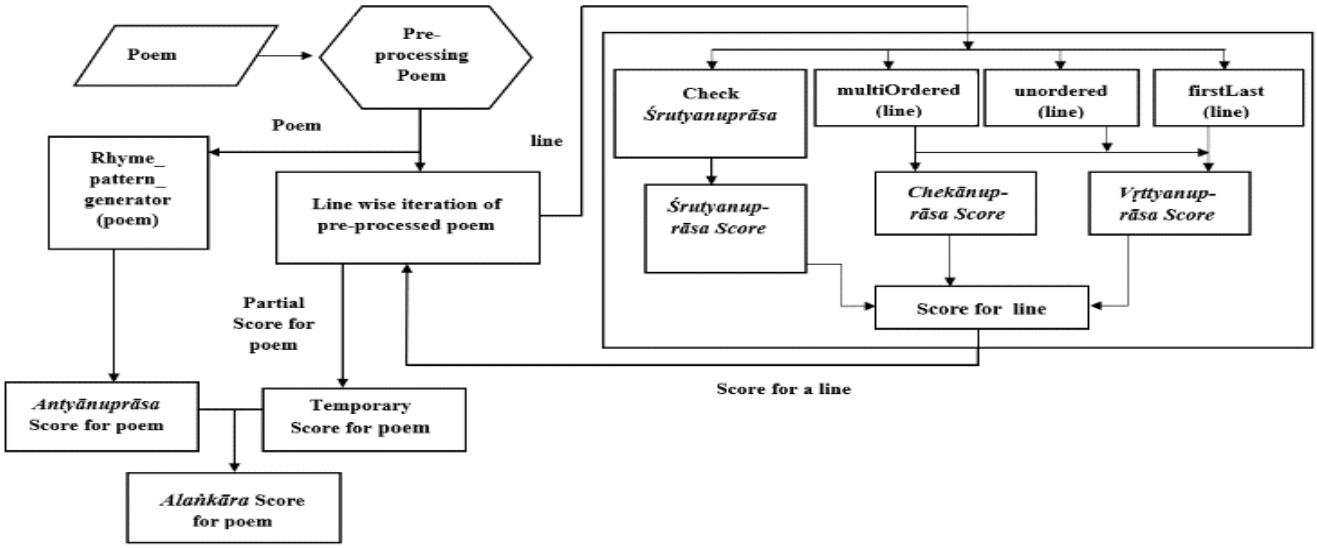

FIGURE 7. Flow diagram for *alaṅkāraScore*.

TABLE 5. Sample input-output from *Text2Mātrā* tool.

Sl. No.	*Chanda* type	Example Input	Output
1	*Dohā*	कबिरा खड़ा बजार में, लिये लुकाठी हाथ । [*kabirā khadā bajāra meṃ, liye lukāṭhī hātha*] जो घर जारै आपनो, चलै हमारे साथ ॥ [*jo ghara jārai āpano, calai hamāre sātha*]	[1121212121212221] [211222121212221]
2	*Caupāī*	छाता छाता कैसा छाता | [*chātā chātā kaisā chātā*] बादल जैसा काला छाता || [*bādala jaisā kālā chātā*] आरे बादल काले बादल | [*āre bādala kāle bādala*] गर्मी दूर भगा रे बादल || [*garmī dūra bhagā re bādala*]	[22222222] [211222222] [2221122211] [2221122211]
3	*Śloka*	तेरा दिल बसेरा हो, घरौंदा प्यार भाव का | [*terā dila baserā ho, gharaundā pyāra bhāva kā*] धर्म सर्वसमाही हो, कर्म धारे विशालता || [*dharma sarvasamāhī ho, karma dhāre viśālatā*]	[2211122212221212] [2121122221221212]
4	*Bhujaṅgaprayāta*	यहां भूख से कौन जीता कभी है [*yahāṃ bhūkha se kauna jītā kabhī hai*] बिके जो बनाया, घरौंदा तभी है [*bike jo banāyā, gharaundā tabhī hai*] तभी तो उजाला, तभी है सवेरा [*tabhī to ujālā, tabhī hai saverā*] तभी बालबच्चे, तभी हाटडेरा [*tabhī bālabacce, tabhī hāṭaḍerā*]	[122122122122] [122122122122] [122122122122] [122122122122]

second is a rhyming pattern in poem corresponding to line 1 and third is the rhyming pattern in the poem corresponding to line 2.

Poem in example 4 (Table 6) is a combination of *PT2* (example 2, Table 6) and *PT1* (example 3, Table 6) for which the output given by the RPaGen is *unknown type,* which is correct as no such poem type is defined. Similarly, for the poem in example 5 (Table 6), output given by the RPaGen is *unknown type,* which is valid as there is no rhyme in the poem.

TABLE 6. Sample input-output from *RPaGen* tool.

Sl. No.	Input Description	Input	Output
1	Poem by Dushyant Kumar	देख, दहलीज से काई नहीं जाने वाली [*dekha, dahalīja se kāī nahīṃ jāne vālī*] ये खतरनाक सचाई नहीं जाने वाली [*ye khataranāka sacāī nahīṃ jāne vālī*] कितना अच्छा है कि सांसों की हवा लगती है [*kitanā acchā hai ki sāṃsoṃ kī havā lagatī hai*] आग अब उनसे बुझाई नहीं जाने वाली [*āga aba unase bujhāī nahīṃ jāne vālī*] एक तालाब-सी भर जाती है हर बारिश में [*eka tālāba-sī bhara jātī hai hara bāriśa meṃ*] मैं समझता हूं ये खाई नहीं जाने वाली [*maiṃ samajhatā hūṃ ye khāī nahīṃ jāne vālī*] चीख निकली तो है होंठों से मगर मद्धम है [*cīkha nikalī to hai hoṇṭhoṃ se magara maddhama hai*] बंद कमरों को सुनाई नहीं जाने वाली [*banda kamaroṃ ko sunāī nahīṃ jāne vālī*]	Poem Rhyming type is 3 [[‘RP3’,’Best’], [[‘RP3’,’Best’], [‘RP3’,’Best’], [‘RP3’,’Best’]], [[‘RP3’,’Medium’], [‘RP3’,’Best’], [‘RP3’,’Medium’]]]
2	A stanza from Dinkar’s poem मंगल-आह्वान [*maṃgala-āhvāna*]	कहते उर के बांध तोड़ [*kahate ura ke bāndha toḍa*] स्वर स्रोत्तों में बह बह अनजान [*svara strottoṃ meṃ baha baha anajāna*] तृण तरु लता अनिल जल थल को [*tṛṇa taru latā anila jala thala ko*] छा लेंगे हम बनकर गान। [*chā leṃge hama banakara gāna।*]	Poem Rhyming type is 2 [[‘RP1’,’Best’]]
3	A *dohā* by Rahim	तैं रहीम मन आपुनो, कीन्हों चारु चकोर। [*taiṃ rahīma mana āpuno, kīnhoṃ cāru cakora*] निसि बासर लागो रहै, कृष्णचंद्र की ओर॥ [*nisi bāsara lāgo rahai, kṛṣṇacandra kī ora*]	Poem Rhyming type is 1 [[‘RP1’,’Best’]]
4	Examples 2 and 3 combined	कहते उर के बांध तोड़ [*kahate ura ke bāndha toḍa*]] स्वर स्रोत्तों में बह बह अनजान, [*svara strottoṃ meṃ baha baha anajāna,*] तृण तरु लता अनिल जल थल को [*tṛṇa taru latā anila jala thala ko*] छा लेंगे हम बनकर गान। [*chā leṃge hama banakara gāna।*] तैं रहीम मन आपुनो, कीन्हों चारु चकोर। [*taiṃ rahīma mana āpuno, kīnhoṃ cāru cakora*] निसि बासर लागो रहै, कृष्णचंद्र की ओर॥ [*nisi bāsara lāgo rahai, kṛṣṇacandra kī ora*]	Unknown Type
5	A *dohā* without rhyme	तैं रहीम मन आपुनो, कीन्हों चारु चकोर। [*taiṃ rahīma mana āpuno, kīnhoṃ cāru*] निसि बासर लागो रहै, कृष्णचंद्र की॥ [*nisi bāsara lāgo rahai, kṛṣṇacandra kī*]	Unknown Type
6	*Dohā*	A file containing 551 *Dohā* by Tulsidas	Poem Rhyming type is 1 [[‘RP1,’Best’], [‘RP1’,’Best’], …]

TABLE 6. *(Continued.)* **Sample input-output from *RPaGen* tool.**

7	*Caupāī*	A file containing 271 *Caupāī* by Tulsidas	Poem Rhyming type is 1 [['RP1','Best'], ['RP1','Best'], ['RP1','Worst], …]
8	Random one line input	देख, दहलीज से काई नहीं जाने वाली [*dekha, dahalīja se kāī nahīṃ jāne vālī*]	Invalid Input

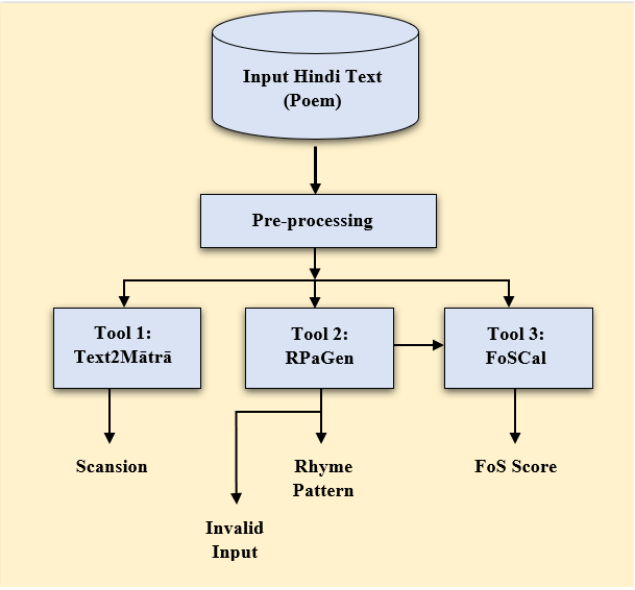

FIGURE 8. Overall contribution of the conveyed research.

In case, the length of the poem supplied as input is not a valid one, RPaGen outputs *invalid input* message (example input 8, Table 6).

C. TOOL 3: FoSCal (FIGURE OF SPEECH CALCULATOR)

FoSCal tool, an implementation of algorithm *Alaṅkāra*_score, measures various rhetoric properties. It then generates a master score for the given input poem. A sufficiently wide spectrum of input types was provided to check the robustness of the tool and some of them are listed below [34], [36].

As discussed, *alaṅkāra* adds beauty and elegance to the poem. A poem is expected to have more *alaṅkāra* than any plain text. FoSCal is tested on 194 stories from *mānasarovara* by Premchand [34] and stories from other authors, FoS score for three of them is displayed in Table 7. Apart from stories, it is tested on 50 newspaper articles, three of them are shown in Table 7. The tool is then used to generate score for 551 *dohā* by Tulsidas (score shown in Table 7), 899 *dohā* by Kabir and poems by Dinkar, Dushyant Kumar & Bachchan. The FoS score generated by the tool stands in favour of the fact stated earlier. The overall contribution of the conveyed research can be viewed in Fig. 8.

TABLE 7. Sample input-output from *FoSCal* tool.

Sample No.	Input	Output (*Alaṅkāra* Score)
1	A file containing 551 dohā	0.733
2	Newspaper Text (Sports section)	0.229286
3	Newspaper Text (Political Section)	0.251462
4	Newspaper Text (Editorial section)	0.233072
5	Story 1	0.235400
6	Story 2	0.235650
7	Story 3	0.244879

There are tools developed for other languages such as Sanskrit metre identifier [37]. This tool outputs the probable metre for any input verse in Sanskrit. The tool is raw and primitive but the work is under progress. *Geet Gatiroop* provides a tool that calculates the instant counts for lines of any input verse in Hindi language [35]. The optimal automated rhyme determining tool is a state of art for Russian poetry [30]. RhymeDesign, is an open-source implementation tool for detecting sonic device in poetry for English language [22].

V. CONCLUSION

The mathematical and statistical study of the aesthetic elements of poetic compositions is important for many reasons. In this connection, in this research, we have proposed some tools for the study and analysis of the elements of Hindi poetry. The Text2Mātrā tool provides a scansion, which can be the foundation for various observations related to *chanda*, like *chanda* type detection and classification, rhythm determination, correctness verification of *chanda*, etc. The RPaGen tool provides a multifunctional output of rhyme pattern(s) for an input Hindi poem. The salutary of the tool can be seen in the very article, the pattern produced by RPaGen is used by FoSCal for *alaṅkāra* scoring. The FoSCal, the maiden tool for *alaṅkāra* score generation, engineered over one of the so many types of *alaṅkāra*, creates a space and motivation for automation of the related works which are still unexplored.

Remark: The 'International Alphabet of Sanskrit Transliteration' (IAST) scheme is used in romanization of Hindi texts used in this manuscript.

REFERENCES

[1] A. Berkani, A. Holzer, and K. Stoffel, "Pattern matching in meter detection of Arabic classical poetry," in *Proc. IEEE/ACS 17th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Nov. 2020, pp. 1–8.

[2] A. Dong and X. Liu, "A statistical method for constructing tang poet social networks," in *Proc. IEEE 14th Int. Conf. Semantic Comput. (ICSC)*, Feb. 2020, pp. 101–107.

[3] A. Ismail, "Expert system for testing the harmony of Arabic poetry," *J. Eng. Sci.*, vol. 1, no. 1, pp. 401–411, 2010.

[4] B. Abuata and A. Al-Omari, "A rule-based algorithm for the detection of Arud meter in CLASSICAL Arabic poetry," *Int. Arab J. Inf. Technol.*, vol. 15, no. 4, pp. 1–5, 2018.

[5] B. K. Joshi and K. K. Kushwah, "A novel approach to automatic detection of Chaupai Chhand in Hindi poems," in *Proc. Int. Conf. Comput., Power Commun. Technol. (GUCON)*, Sep. 2018, pp. 223–228.

[6] B. Nagy, "Metre as a stylometric feature in Latin hexameter poetry," *Digit. Scholarship Humanities*, vol. 36, no. 4, pp. 999–1012, Oct. 2021.

[7] D. N. Sharma, "Alankaar," in *Kavya Ke Tatva*, 1st ed. Allahabad, India: Lokbharti Prakashan, 2021.

[8] J. Kaur and J. R. Saini, "Automatic classification of Punjabi poetries using poetic features," *Int. J. Comput. Intell. Stud.*, vol. 7, no. 2, pp. 124–137, 2018.

[9] J. Parakh, "The whole world is a family—Modern Hindi poetry and the larger cause of humanity," *Proc. Social Behav. Sci.*, vol. 2, no. 5, pp. 7435–7445, 2010.

[10] J. Pradhan and S. Chakraborty, "'The road not taken' statistically taken: A probabilistic analysis of Frost's poem," *J. Statist. Manage. Syst.*, vol. 24, no. 8, pp. 1733–1752, Nov. 2021.

[11] K. D. Dvivedi and S. L. Singh, "Chapter 1," in *The Prosody Pingala*, 2nd ed. Varanasi, India: Vishwavidyalaya Prakashan, 2013.

[12] K. K. Kushwah and B. K. Joshi, "Rola: An Equi–Matrik Chhand of Hindi poems," *Int. J. Comput. Sci. Inf. Secur.*, vol. 15, no. 3, pp. 362–364, 2017.

[13] L. Koss, "Differential equations in literature, poetry and film," *J. Math. Arts*, vol. 9, nos. 1–2, pp. 1–16, Apr. 2015.

[14] H. Meirong, J. Huimin, and Y. Yangrui, "Study on the rhythm of Tibetan poems based on breathing signal," in *Proc. Int. Conf. Intell. Comput. Technol. Autom.*, May 2010, pp. 618–621.

[15] M. K. Audichya and J. R. Saini, "Computational linguistic prosody rule-based unified technique for automatic metadata generation for Hindi poetry," in *Proc. 1st Int. Conf. Adv. Inf. Technol. (ICAIT)*, Jul. 2019, pp. 436–442.

[16] M. K. Audichya and J. R. Saini, "Stanza type identification using systematization of versification system of Hindi poetry," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 1, pp. 142–153, 2021.

[17] M. K. Audichya and J. R. Saini, "Towards natural language processing with figures of speech in Hindi poetry," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 3, pp. 128–133, 2021.

[18] M. S. Al-Shaibani, Z. Alyafeai, and I. Ahmad, "Meter classification of Arabic poems using deep bidirectional recurrent neural networks," *Pattern Recognit. Lett.*, vol. 136, pp. 1–7, Aug. 2020.

[19] N. Das, "Khanda-1: Samaanya chanda charcha," in *Hindi Chhandolakshan*, 2nd ed. Delhi, India: Vani Prakashan, 2005.

[20] N. Ghneim and O. Alsharif, "Emotion classification in Arabic poetry using machine learning," *Int. J. Comput. Appl.*, vol. 65, no. 16, pp. 10–15, 2013.

[21] N. K. Singh, S. Chakraborty, and M. Roy, "Analysing the poetic structure of Jana-Gana-Mana in entirety: A statistical approach," *J. Appl. Math. Comput.*, vol. 5, no. 4, pp. 264–272, Oct. 2021.

[22] N. McCurdy, V. Srikumar, and M. Meyer, "RhymeDesign: A tool for analyzing sonic devices in poetry," in *Proc. 4th Workshop Comput. Linguistics Literature*, Jun. 2015, pp. 12–22.

[23] N. Swami, "Alankaar," in *Alankaar Parijaat*, 18th ed. Agra, India: Lakshmi Narayan Agrawal, 2022.

[24] P. B. Bafna and J. R. Saini, "On exhaustive evaluation of eager machine learning algorithms for classification of Hindi verses," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 2, pp. 181–185, 2020.

[25] R. Aharoni, "Mathematics, poetry and beauty," *J. Math. Arts*, vol. 8, nos. 1–2, pp. 5–12, 2014.

[26] R. Deshmukh, "Marathi poem classification using machine learning," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2, pp. 2723–2727, 2019.

[27] R. K. Agnihotri, "Part VII: Sounds and script," in *Hindi: An Essential Grammar*, 1st ed. New York, NY, USA: Routledge, 2007.

[28] S. Hamidi, F. Razzazi, and M. P. Ghaemmaghami, "Automatic meter classification in Persian poetries using support vector machines," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol. (ISSPIT)*, Dec. 2009, pp. 563–567.

[29] S. Pandey, *Chanda Manjari*. Allahabad, India: Anjuman Prakashan, 2015.

[30] V. Barakhnin, I. Kuznetsova, O. Kozhemyakina, Y. Borzilova, and I. Pastushkov, "Improvement of the algorithm of automated definition of rhyme," in *Proc. VI Int. Conf. Inf. Technol. Nanotechnol. (ITNT)*, 2020, pp. 36–41.

[31] V. Kumar, "Varn-Vichaar," in *Vrihat Vyakarana Bhaskar*, 2nd ed. Uttar Pradesh, India: Bharati Bhawan, 2018.

[32] Y. Xu, "Differentiated prosodic adaption of Chinese and English poetry: An acoustic approach to reading of Chinese tang poetry and Shakespearean sonnets," in *Proc. 12th Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC), Virtual Mode*, Dec. 2020, pp. 211–215.

[33] (2011). *Census of India, Language: India, States and Union Territories*. Accessed: Feb. 2022. [Online]. Available: https://censusindia.gov.in/2011Census/C-16_25062018_NEW.pdf

[34] (2015). *[Dataset] Gadyakosh*. Accessed: Jan. 2022. [Online]. Available: http://gadyakosh.org/

[35] (Jun. 2022). *Geet Gatiroop, Quantity Calculation*. Consulted. [Online]. Available: https://geet-gatiroop.manaskriti.com/

[36] (2018). *[Dataset] Kavitakosh*. Consulted. Accessed: Feb. 2022. [Online]. Available: http://kavitakosh.org/kk

[37] (2018). *Sanskrit Metres*. Consulted. Accessed: Jun. 2022. [Online]. Available: http://sanskritmetres.appspot.com/

KOMAL NAAZ was born in Jharkhand, India, in 1994. She received the B.Tech. degree in computer science and engineering from the Kalinga Institute of Technology, Bhubaneswar, India, in 2017, and the M.Tech. degree in computer science and engineering from the Birla Institute of Technology Mesra, Mesra, India, in 2021, where she is currently pursuing the Ph.D. degree in engineering.

NIRAJ KUMAR SINGH was born in Jharkhand, India, in 1981. He received the B.E. degree in computer science and engineering from VTU, Belgaum, India, in 2004, and the M.E. degree in computer science and engineering and the Ph.D. degree in engineering from the Birla Institute of Technology Mesra, Mesra, India, in 2012 and 2017, respectively.

He is currently an Assistant Professor with the Birla Institute of Technology Mesra, since 2016. His research interests include design and analysis of computer experiments and computational analysis of poetry.