A user-friendly tool for metrical analysis of Sanskrit verse

Shreevatsa Rajagopalan

Abstract: This paper describes the design and implementation of a tool that assists readers of metrical verse in Sanskrit (and other languages/literatures with similar prosody). It is open-source, and available online as a web application, as a command-line tool and as a software library. It handles both *varṇavṛtta* and *mātrāvṛtta* metres. It has many features for usability without placing strict demands on its users. These include allowing input in a wide variety of transliteration schemes, being fairly robust against typographical or metrical errors in the input, and "aligning" the given verse in light of the recognized metre.

This paper describes the various components of the system and its user interface, and details of interest such as the heuristics used in the identifier and the dynamic-programming algorithm used for displaying results. Although originally and primarily designed to help readers, the tool can also be used for additional applications such as detecting metrical errors in digital texts (its very first version identified 23 errors in a Sanskrit text from an online corpus), and generating statistics about metres found in a larger text or corpus. These applications are illustrated here, along with plans for future improvements.

1 Introduction

1.1 Demo

As a software tool is being discussed, it seems best to start with a demonstration of a potential user interaction with the tool. Suppose I wish to learn about the metre of the following *subhāṣita* (which occurs in the *Pratijñāyaugandharāyaṇa* attributed to Bhāsa):

kāṣṭhād agnir jāyate mathya-mānād-
bhūmis toyaṃ khanya-mānā dadāti |
sotsāhānāṃ nāstyasādhyaṃ narāṇāṃ
mārgārabdhāḥ sarva-yatnāḥ phalanti ||

Then I can visit the tool's website, `http://sanskritmetres.appspot.com`, enter the above verse (exactly as above), and correctly learn that it is in the metre *Śālinī*. More interestingly, suppose I do not have the verse correctly: perhaps I am quoting it from memory (possibly having misheard it, and unaware of the line breaks), or I have found the text on a (not very reliable) blog, or some errors have crept into the digital text, or possibly I just make some mistakes while typing. In such a case, possibly even with an unreasonable number of mistakes present, I can still use the tool in the same way. Thus, I can enter the following error-ridden input (which, for illustration, is encoded this time in the ITRANS convention):

```
kaaShThaad agni jaayate
mathyamaanaad bhuumistoya khanyamaanaa /
daati sotsaahaanaaM naastyasaadhyaM
naraaNaaM maargaabdhaaH savayatnaaH phalantiihi //
```

Here, some syllables have the wrong prosodic weight (*laghu* instead of *guru* and vice-versa), some syllables are missing, some have been introduced extraneously, not a single line of the input is correct, and even the total number of syllables is wrong. Despite this, the tool identifies the metre as Śālinī. The output from the tool, indicating the identified metre, and highlighting the extent to which the given verse corresponds to that metre, is shown in Figure 1. The rest of this paper explains how this is done, among other things.

1.2 Background

A large part of Sanskrit literature, in *kāvya*, *śāstra* and other genres, is in verse (*padya*) rather than prose (*gadya*). A verse in Sanskrit (not counting some modern Sanskrit poets' experimentation with "free verse" and the like) is invariably in metre.

Computer tools to recognize the metre of a Sanskrit verse are not new. A script in the Perl programming language, called `sscan`, written by John Smith, is distributed among other utilities at the `http://bombay.`

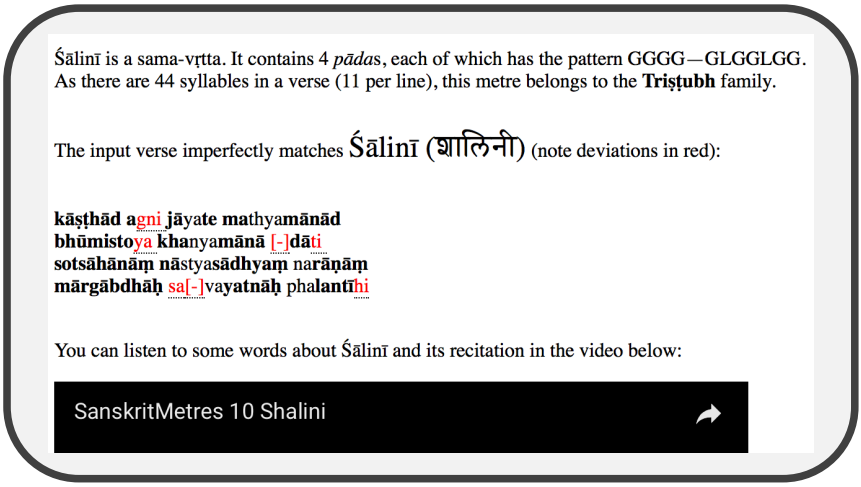

Figure 1

A screenshot of the output from the tool for highly erroneous input. Despite the errors, the metre is correctly identified as Śālinī. The guru syllables are marked in bold, and the deviations from the expected metrical pattern (syllables with the wrong weight, or missing or superfluous syllables) are underlined (and highlighted in red).

indology.info website, and although the exact date is unknown, the timestamp in the ZIP file suggests a date of 1998 or earlier for this file (Smith 1998?). In fact, this script, only 61 lines long (38 source lines not including comments and description) was the spark of inspiration that initiated the writing of the tool being described in the current paper, in 2013. Other software or programs include those by Murthy (2003?), by A. Mishra (2007) and by Melnad, Goyal, and P. M. Scharf (2015). A general introduction to metre and to Sanskrit prosody is omitted in this paper for reasons of space, as the last of these papers (Melnad, Goyal, and P. M. Scharf 2015) quite excellently covers the topic.

Like these other tools, the tool being discussed in this paper recognizes the metre given a Sanskrit verse. It is available in several forms: as a web application hosted online at http://sanskrit-metres.appspot.com, as a commandline tool, and as a Python library; all are available in source-code form at https://github.com/shreevatsa/sanskrit. It is being described here for two reasons:

1. It has some new features that I think will be interesting (see section 1.4), some of which distinguish it from other tools. The development of this tool has thrown up a few insights (see Section 4) which may be useful to others who would like to develop better tools in future.

2. A question was raised about this tool (P. Scharf 2016), namely:

   > "An open source web archive of metrically related software and data can be found at https://github.com/shreevatsa/sanskrit with an interface at http://sanskritmetres.appspot.com/. The author and contributors to this archive and data were unknown at the time and not included in our literature review. No description of the extent, comprehensiveness, and effectiveness of the software has been found."

   I took this as encouragement that such a description may be desirable / of interest to others.

1.3 The intended user

The tool can be useful for someone trying to read or compose Sanskrit verses, and for someone checking a text for metrical errors. In other words, the tool

can be used by different kinds of users: a curious learner, an editor working with a text (checking verses for metrical correctness), a scholar investigating the metrical signature of a text, or an aspiring poet. To make these concrete, consider the following "user stories" as motivating examples.

- **Devadatta** is learning Sanskrit. He knows that Sanskrit verse is written in metre and that this is supposed to make it easier to chant or recite. But he knows very little about various metres, so that when he looks at a verse, especially one in a longer metre like *Śārdūla-vikrīḍitam* or *Sragdharā*, he cannot quickly recognize the metre. All he sees is a string of syllables, and he has no idea where to pause (*yati*), how to recite, or even where to break at *pādā*s if they are not indicated clearly in the text he is reading. With this tool, these problems are solved, and he can focus on understanding and appreciating the poetry, now that he can read it aloud in a rhythmic and melodic way and savour its sounds.

- **Chitralekha** is a scholar. She works with digital texts that, though useful to have, are sometimes of questionable provenance and do not always meet her standards of critical editions. Errors might have crept into the texts, and she has the idea that some instances of scribal emendation or typographical errors (such as omitted letters, extraneous letters, or transposed letters) are likely to cause metrical errors as well. With this tool, she can catch a great many of them (see Section 3). Sometimes, she is interested in questions about prosody itself, such as: what are all the metres used in this text? Which ones are the most common? How frequently does the poet X use a particular "poetic licence" of scansion? What are the rules governing *Anuṣṭubh* (*Śloka*), typically? This tool can help her with such questions too.

- **Kamban** would like to write poetry, like his famous namesake. He has a good command of vocabulary and grammar, and has some poetic imagination, but when he writes a verse, especially in an "exotic" (to him) metre, he is sometimes unsure whether he has got all the syllables right. With this tool, he enters his tentative attempt, and sees whether anything is off. He knows that the metres will soon become second-nature to him and he will not need the tool anymore, but still he

wishes he could have more help—such as choosing his desired metre, and knowing what he needs to add to his partially composed verse.

With the names of these users as mnemonics, we can say that the tool can be used to *discover*, *check*, and *compose* metrical verse and facts about them.

1.4 User-friendly features

As mentioned earlier, the tool has a number of features for easing the user's job:

1. It accepts a wide variety of input scripts (transliteration schemes). Unlike most tools, it does not enforce the input to be in any particular input scheme or system of transliteration. Instead, it accepts IAST, Harvard-Kyoto, and ITRANS transliteration, Unicode Devanāgarī and Unicode Kannada scripts, without the user having to indicate which input scheme is used. The tool is agnostic to the input method used, as it converts all input to an internal representation based on SLP1 (P. M. Scharf and Hyman 2011). It is straightforward to extend to other scripts or transliteration methods, such as SLP1 or other Indic scripts.

2. It is highly robust against typographical errors or metrical errors in the verse that is input. This is perhaps the most interesting feature of the tool, and is useful because text in the "wild" is not always error-free.

3. It can detect the metre even from partial verses—even if the user is not aware that the verse one is looking up is incomplete.

4. Informative "display" of a verse in relation to the identified metre, by aligning the verse to the metre using a dynamic programming algorithm to find the best alignment.

5. Supports learning more about a metre, by pointing to other examples of the metre, and audio recordings of the metre being recited in several styles (where available).

6. Quick link to provide feedback (by creating an issue on GitHub), specific to the input verse being processed on the page.

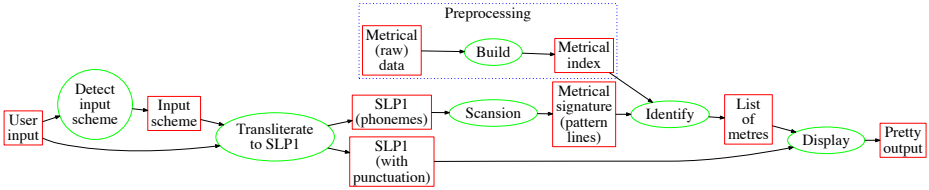

Figure 2

A "data flow diagram" of the system's operation. The rectangles denote different forms taken by the data; the ovals denote code that transforms (or uses, or generates) the data.

2 How it works

This section describes how the system works. At a high level, there are the following steps/components:

1. Metrical data, about many known metres. This has been entered into the system.

2. Building the Index (Pre-processing): from the metrical data, various indices are generated.

3. Detection and Transliteration: The input supplied by the user is examined, the input scheme detected, and transliterated into SLP1.

4. Scansion: The SLP1 text (denoting a set of phonemes) is translated into a metrical signature (a pattern of *laghu*s and *guru*s).

5. Matching: The metrical signature is compared against the index, to identify the metre (or metres).

6. Display: The user's input is displayed to the user, appropriately re-formatted to fit the identified metre(s) and with highlighting of any deviations from the metrical ideal.

These steps are depicted in Figure 2, and described in more detail in the following subsections.

2.1 Metrical data

This is the raw data about all the metres known to the system. They are stored in the JSON format, so that they could be used by other programs too. In what follows, a metrical **pattern** is defined as string over the alphabet $\{\mathsf{L}, \mathsf{G}\}$, i.e., a sequence of symbols each of which is either L (denoting *laghu* or a light syllable) or G (denoting *guru* or a heavy syllable). As described elsewhere (Melnad, Goyal, and P. M. Scharf 2015), there are two main types of metres, *varṇavṛtta* and *mātrāvṛtta* (note that (Murthy 2003) points out that the *Śloka* metre constitutes a third type by itself), with the former having three subtypes:

1. *samavṛtta* metres, in which all four *pāda*s of a verse have the same metrical pattern,
2. *ardhasamavṛtta* metres, in which odd *pāda*s have one pattern and even *pāda*s another (so that the two halves of the verse have the same metrical pattern),
3. *viṣamavṛtta* metres, in which potentially all four *pāda*s have different metrical patterns.

Correspondingly each metre's characteristics are indicated in this system with the minimal amount of data necessary:

1. *samavṛtta* metres are represented by a list of length one (or for convenience, simply a string), containing the pattern of each of their *pāda*s,
2. *ardhasamavṛtta* metres are represented by a list of length two, containing the pattern of the odd *pāda*s followed by the pattern of the even *pāda*s,
3. *viṣamavṛtta* metres are represented by a list of length four, containing the pattern for each of the four *pāa*s.

Additionally, with the pattern, *yati* can be indicated; also spaces can be added, which are ignored. The yati is ignored for identification, but used later for displaying information about the metre. Here are some lines, as examples:

```
{
        # ...
        ['Śālinī', 'GGGG-GLGGLGG'],
        ['Praharṣiṇī', 'GGGLLLLGLGLGG'],
        ['Bhujaṅgaprayātam', 'LGG LGG LGG LGG'],
        # ...
        ['Viyoginī', ['LLGLLGLGLG','LLGGLLGLGLG']],
        # ...
        ['Udgatā', ['LLGLGLLLGL',
                    'LLLLLGLGLG',
                    'GLLLLLLGLLG',
                    'LLGLGLLLGLGLG']],
        # ...
}
```

For *mātrāvṛtta* metres (those based on the number of morae: *mātrā*s), the constraints are more subtle, and as not every syllable's weight is fixed, there are so many patterns that fit each metre that it may not be efficient to generate and store each pattern separately. Instead, the system represents them by using a certain conventional notation, which expands to regular expressions. This notation is inspired by the elegant notation described in another paper (Melnad, Goyal, and P. M. Scharf 2015), and uses a particularly useful description of the Āryā and related metres available in a paper by Ollett Ollett (2012).

```
    # ...
["Āryā", ["22 4 22", "4 22 121 22 .", "22 4 22", "4 22 1 22 ."]],
    ["Gīti", ["22 4 22", "4 22 121 22 ."]],
    ["Upagīti", ["22 4 22", "4 22 L 22 ."]],
["Udgīti", ["22 4 22", "4 22 L 22 .", "22 4 22", "4 22 121 22."]],
    ["Āryāgīti", [["22 4 22", "4 22 121 22 (4|2L)"]],
    # ...
```

Here, `2` will be interpreted as the regular expression `(G|LL)` and `4` as the regular expression `(GG|LLG|GLL|LLLL|LGL)` – all possible sequences of *laghu*s and *guru*s that are exactly 4 *mātrā*s long. Note that with this notation, the frequently mentioned rule of "any combination of 4 *mātrā*s except `LGL` (*ja-gaṇa*)" is simply denoted as `22`, expanding to the regular expression `(G|LL)(G|LL)` which covers precisely the 4 sequences of *laghu*s and *guru*s of total duration 4, other than `LGL`.

Type of metre	Number
samavṛtta	1242
ardhasamavṛtta	132
viṣamavṛtta	19
mātrāvṛtta	5
Total	1398

Table 1

The number of metres "known" to the current system. Not too much should be read into the raw numbers as a larger number isn't necessarily better; see Section 4.1.3 for why.

The data in the system was started with a hand-curated list of popular metres (Ganesh 2013). It was greatly extended with the contributions of Dhaval Patel, which drew from the *Vṛttaratnākara* and the work of Mishra (A. Mishra 2007). A few metres from these contributions are yet to be incorporated, because of reasons described in section 4.1.3. Overall, as a result of all this, at the moment we have a large number of known metres, shown in Table 1.

2.2 Metrical index

The data described in the previous section is not used directly by the rest of the program. Instead, it is first processed into data structures (which we can consider a sort of "index") that allow for efficient lookup, even when the number of metres is huge. These enable the robustness to errors that is one of the most important features of the system. The indices are called PĀDA1, PĀDA2, PĀDA3, PĀDA4, ARDHA1, ARDHA2, and FULL. Each of these indices consists of an associative array (a Python `dict`) that maps a pattern (a "pattern" is a string over the alphabet {L, G}) to a list[1] of metres that contain that pattern (at the position indicated by the name of the index), and similarly an array that maps a regular expression to the list of metres that contain it. For instance, `ardha2` maps the second half of each known metre to that metre's name. It is at this point that we also introduce *laghu-*

[1]Why a list? Because different metres can share the same *pāda*, for instance. And there can even be multiple names for the same metre. See Section 4.1.3 later.

ending variants for many metres (see more in 4.1.2). Section 2.5 describes how these indices are used.

Although this index is generated automatically and not written down in code, the following hypothetical code illustrates some sample entries in the `ardha2` index:

```
ardha2_patterns = {
    # ...
    'GGGGGLGGLGGGGGGGLGGLGG': ['Śālinī'],
    # laghu variants for illustration.
    # In reality we don't add for Śālinī…
    'GGGGGLGGLGLGGGGGLGGLGG': ['Śālinī'],
    'GGGGGLGGLGGGGGGGLGGLGL': ['Śālinī'],
    'GGGGGLGGLGLGGGGGLGGLGL': ['Śālinī'],
    # ...
}
ardha2_regexes = {
    # ...
    "22 4 22" + "4 22 L 22 .": ['Āryā', 'Upagīti],
    # ...
}
```

2.3 Transliteration

The first step that happens after users enter their input is automatic transliteration. Detecting the input scheme is based on a few heuristics. Among the input schemes initially supported (Devanāgarī, Kannada, IAST, ITRANS, and Harvard-Kyoto), the detection is done as follows:

- If the input contains any Kannada consonants and vowels, treat it as Kannada.
- If the input contains (m)any Devanāgarī consonants and vowels, treat it as Devanāgarī. Note that this should *not* be applied to other characters from the Devanāgarī Unicode block, such the *daṇḍa* symbol, which are often used with other scripts too, as encouraged in the Unicode standard.
- If the input contains any of the characters ĀĪŪṚṜḶḸṂḤṄÑṬḌŚṢ, treat it as IAST.

- If the input matches the regular expression

  ```
  aa|ii|uu|[RrLl]\^[Ii]|RR[Ii]|LL[Ii]|~N|Ch|~n|N\^|Sh|sh
  ```

  treat it as ITRANS. Here, the `Sh` and `sh` might seem dangerous, but the consonant cluster श्ह is unlikely in Sanskrit.

- Else, treat the input as Harvard-Kyoto.

An option to explicitly indicate the input scheme (bypassing the automatic inference) could be added, but has not seemed necessary so far. The input is transliterated into (a subset of) the encoding SLP1 (P. M. Scharf and Hyman 2011), which is used internally, as it is has many properties suitable for computer representation of Sanskrit text. While the input is being transliterated according to the detected scheme, known punctuation marks (and line breaks) are retained, while all "unknown" characters that have not been programmed into the transliterator (such as control characters and accent marks in Devanāgarī) are ignored.

The exact details of how the transliteration is done are omitted here, as transliteration may be regarded as a reasonably well-solved problem by now. One point worth mentioning is that there are no strict input conventions. In other work (Melnad, Goyal, and P. M. Scharf 2015), a convention is adopted like:

> If the input text lacks line-end markers, it is assumed to be a single pāda and to belong to the samavṛtta type of meter

Such a scheme may be interesting to explore. For now, as much as possible, the system tries to assume an untrained user and therefore infer all such things, or try all possibilities.

2.4 Scan

The transliteration into SLP1 can be thought of as having generated a set of Sanskrit phonemes (this close relationship between phonemes and the textual representation is the primary strength of the SLP1 encoding). From these phonemes, scansion into a pattern of *laghu*s and *guru*s can proceed directly, without bothering with syllabification (however, syllabification is still done, for the sake of the "alignment" described later in section 2.6). The rule for scansion is mechanical: initial consonants are dropped, and

each vowel is considered as a set along with all the non-vowels that follow it before the next vowel (or end of text) is found. If the vowel is long or if there are multiple consonants (treating *anusvāra* and *visarga* as consonants here, for the purpose of scansion only) in this set, then we have a *guru*, else we have a *laghu*.

The validity of this method of scansion, with reference to the traditional Sanskrit grammatical and metrical texts is skipped in this paper, as something similar has been treated elsewhere (Melnad, Goyal, and P. M. Scharf 2015). However, note that this is the "purist" version of Sanskrit scansion. There is an issue of *śithila-dvitva* or poetic licence, which is treated in more detail in Section 4.3.

2.5 Identification

The core of the tool's robust metre identification is an algorithm for trying many possibilities for identifying the metre of the input text. Identifying the metre given a metrical pattern (result of scansion) is done in two steps: (1) first the input is broken into several "parts" in various ways, and then (2) each of these parts is matched against the appropriate indices.

2.5.1 Parts

Given the metrical pattern corresponding to the input text, which may be either a full verse, a half-verse or a single quarter-verse (*pāda*), we try to break it into parts in multiple ways. One way of breaking the input, which should not be ignored, is already given by the user, in the form of line breaks in the input. If there are 4 lines for example, it is a strong possibility that these are the 4 *pāda*s of the verse. If there are 2 lines, it is possible that each line contains two *pāda*s. But what if there are 3 lines, or 5? Another way of breaking the input is by counting syllables. If the number of syllables is a multiple of 4 (say $4n$), it is possible that every n syllables constitute a *pāda* of a *samavṛtta* metre. But what if the number of syllables is not a multiple of 4?

The solution adopted here is to consider all ways of breaking a pattern into k parts even when its length (say l) may not be a multiple of k. Although this would apply to any positive k, we only care about $k = 4$ and $k = 2$, so let's focus on the $k = 4$ case for illustration. In that case, suppose that the

length l leaves a remainder r when divided by 4, that is,

$$l \equiv r \pmod 4, \quad 0 \leq r < 4$$

or in other words l can be written as $l = 4n + r$ for some integer n, where $0 \leq r < 4$. Then, as $\lfloor l/4 \rfloor = n$ (here $\lfloor \cdot \rfloor$ denotes the "floor function", or integer division with rounding down), we can consider all the ways of breaking the string of length l into 4 parts of lengths $(n+a, n+b, n+c, n+d)$ where $a + b + c + d = r$ (in words: we consider all ways of distributing the remainder r among the 4 parts). For example, when $r = 2$, we say that a string of length $4n + 2$ can be broken into 4 parts in 10 ways:

$$\begin{gathered}
(n, n, n, n+2) \\
(n, n, n+1, n+1) \\
(n, n, n+2, n) \\
(n, n+1, n, n+1) \\
(n, n+1, n+1, n) \\
(n, n+2, n, n) \\
(n+1, n, n, n+1) \\
(n+1, n, n+1, n) \\
(n+1, n+1, n, n) \\
(n+2, n, n, n)
\end{gathered}$$

Similarly, there are 4 ways when $r = 1$, 20 ways when $r = 3$, and of course there is exactly one way (n, n, n, n) when $r = 0$.

In this way, we can break the given string into 4 parts (in 1, 4, 10, or 20 ways) or into 2 parts (in 1 or 2 ways), *either by lines or by syllables*. For instance, if we are given an input of 5 lines, then there are 4 ways we can break it into 4 parts, by lines. What we do with these parts is explained next.

2.5.2 Lookup/match

Once we have the input broken into the appropriate number of parts (based on whether we're treating it as a full verse, a half verse, or a *pāda*), we look up each part in the appropriate index. For a particular index, to match against patterns is a direct lookup (we do not have to loop through all patterns in

	treating input as		
kind of index	full verse	half verse	single *pāda*
pāda1	first part of 4	first part of 2	the full input
pāda2	second part of 4	second part of 2	the full input
pāda3	third part of 4	first part of 2	the full input
pāda4	fourth part of 4	second part of 2	the full input
ardha1	first part of 2	the full input	-
ardha2	second part of 2	the full input	-
full	the full input	-	-

Table 2

What to match or look up, depending on how the input is being treated. Everywhere in the table above, phrases like "first part of 4*" mean both by lines and by syllables. For instance, when treating the input as a full verse, the first* $^1/_4$ *part by lines and the first* $^1/_4$ *part by syllables are* both *matched against the* `pāda1` *index.*

the index). To match against regexes, we do indeed loop through all regexes, which are fewer in number compared to the number of patterns. If needed, we can trade-off time and memory here; for instance we could have indexed a large number of instantiated patterns instead of regexes even for *mātrā* metres. Note that in this way, to match an *ardhasamavṛtta* or a *viṣamavṛtta* that has been input perfectly, we search directly for the full pattern (of the entire verse) in the index. We do not have to run a loop for breaking a line into *pāda*s in all possible ways, as in (Melnad, Goyal, and P. M. Scharf 2015). Details of which indices are looked up are in Table 2.

2.6 Align/Display

The metre identifier, from the previous section, results in a list of metres that are potential matches to the input text. Not all of them may match the input verse perfectly; some may have been detected on the basis of partial matches. Whatever the reason for this imperfect match (an over-eager matching on the part of the metre identifier, or errors in the input text), it would be useful for the user to see how closely their input matches a given metre. And even when the match is perfect, aligning the verse to the metre can help highlight the *pāda* breaks, the location of *yati*, and so on.

This is done by the tool, using a simple dynamic-programming algorithm very similar to the standard algorithm for the longest common subsequence problem: in effect, we simply align both the strings (the metrical pattern of the input verse, and that of the known metre) along their longest common subsequence.

What this means is that given two strings `s` and `t`, we use a dynamic programming algorithm to find the minimal set of "gap" characters to insert in each string, such that the resulting strings match wherever both have a non-gap character (and never have a gap character in both). For example:

```
('abcab', 'bca'),   =>   ('abcab', '-bca-')
('hello', 'hello'),   =>   ('hello', 'hello')
('hello', 'hell'),   =>   ('hello', 'hell-')
('hello', 'ohell'),   =>   ('-hello', 'ohell-')
('abcdabcd', 'abcd'),   =>   ('abcdabcd', 'abcd----')
('abcab', 'acb'),   =>   ('abcab', 'a-c-b')
('abcab', 'acbd'),   =>   ('abcab-', 'a-c-bd')
```

We use this algorithm on the verse pattern and the metre's pattern, to decide how to align them. Then, using this alignment, we display the user's input verse in its display version (transliterated into IAST, and with some recognized punctuation retained). Here, *laghu* and *guru* syllables are styled differently in the web application (styling customizable with CSS). This also highlights each location of *yati* or caesura (if known and stored for the metre), so that the user can see if their verse violates any of the subtler rules, such as words straddling *yati* boundaries.

This algorithm could also be used for *ranking* the results, based on the degree of match between the input and each result (metre identified).

3 Text analysis and results

As part of testing the tool (and as part of pursuing the interest in literature and prosody that led to the tool in the first place), a large number of texts such as from GRETIL[2] were examined. Although primarily designed to help readers, the tool can also be used to analyze a metrical text, to catch errors or generate statistics about the metres used. In the very first version of the

[2]Göttingen Register of Electronic Texts in Indian Languages: and related Indological materials from Central and Southeast Asia, `http://gretil.sub.uni-goettingen.de`

tool, the first metre added was *Mandākrāntā*, and the tool was run on a text of the *Meghadūta* from GRETIL, the online corpus of Sanskrit texts. This text was chosen because the *Meghadūta* is well-known to be entirely in the Mandākrāntā metre, so the "gold standard" to use as a reference to compare against was straightforward. Surprisingly, this tool successfully identified 23 errors in the 122 verses![3] These were communicated to the GRETIL maintainer.

Similarly, testing of the tool on other texts highlighted many errors. Errors identified in the GRETIL text of Bhartṛhari's *Śatakatraya* were carefully compared against the critical edition by D. D. Kosambi.[4] In this text, as in Nīlakaṇṭha's *Kali-viḍambana*,[5] in Bhallaṭa's *Bhallaṭa-śataka*,[6], and in almost all cases, the testing highlighted errors in the text, rather than any in the metre recognizer. This constitutes evidence that the recognizer has a high accuracy approaching 100%, though the lack of a reliable (and nontrivial) "gold standard" hinders attaching a numeric value to the accuracy. In the terminology of "precision and recall", the recognizer has a recall of 100% in the examples tested (for example, no verse that is properly in Śārdūla-vikrīḍitam is failed to be recognized as that metre), while the precision was lower and harder to measure because of errors in the input (sufficiently many errors can make the verse partially match a different metre).

After sufficiently fixing the tool and the text so that Meghadūta was recognized as being 100% in the *Mandākrāntā* metre, other texts were examined. These statistics[7] confirmed that, for example, the most common metres in the *Amaruśataka* are *Śārdūlavikrīḍitam* (57%), *Hariṇī* (13%) and *Śikhariṇī* (10%), while those in Kālidāsa's *Raghuvamśa* are *Śloka*, *Upajāti* and *Rathoddhatā*. And so on. Once errors in the texts are fixed, this sort of analysis can give insights into the way different poets use metre. It can also

[3] See a list of 23 errors and 3 instances of broken sandhi detected in one of the GRETIL texts of the *Meghadūta*, at `https://github.com/shreevatsa/sanskrit/blob/f2ef7364/meghdk_u_errors.txt` (October 2013).

[4] See `https://github.com/shreevatsa/sanskrit/blob/7c42546/texts/gretil_stats/diff-bharst_u.htm-old.patch` for a list of errors found, in diff format, with comments referring to the location of the verse in Kosambi

[5] `https://github.com/shreevatsa/sanskrit/blob/08ccb91/texts/gretil_stats/diff-nkalivpu.htm.patch`

[6] `https://github.com/shreevatsa/sanskrit/blob/67251bc/texts/gretil_stats/diff-bhall_pu.htm.patch`

[7] `http://sanskritmetres.appspot.com/statistics`

be used for students to know which are the most common metres to focus on learning, at least for a given corpus. Other sources of online texts, like TITUS, SARIT[8] or The Sanskrit Library[9] could also be used for testing the system.

4 Interesting issues and computational experience

Some insights and lessons learned as a result of this project are worth highlighting, as are some of the design decisions that were made either intentionally or unconsciously.

4.1 Metrical data

4.1.1 The gaṇa-s

For representing the characteristics of a given metre, a popular scheme used by all Sanskrit authors of works on prosody is the use of the 8 *gaṇ*s. Each possible *laghu-guru* combination of three syllables (*trika*), namely each of the 2^3 possibilities LLL, LLG, LGL, LGG, GLL, GLG, GGL, GGG, is given a distinct name (*na*, *sa*, *ja*, *ya*, *bha*, *ra*, *ta*, *ma* respectively), so that a long pattern of *laghu*s and *guru*s can be concisely stated in groups of three. This is an excellent mnemonic and space-saving device, akin to writing in octal instead of binary. For instance, the binary number 110110010101_2 can be written more concisely as the octal number 6625_8 and the translation between them is immediately apparent (110110010101_2 corresponds to 6625_8 and vice-versa, by simply treating each group of three binary digits (bits) as an octal digit, or conversely expanding each octal digit to a three-bit representation). Similarly, the pattern GGLGGLLGLGLG of *Indravaṃśa* can be more concisely expressed by the description as "*ta ta ja ra*". Moreover, another mnemonic device of unknown origin uses a string "*yamātārājabhānasalagaṃ*" that traverses all the 8 *gaṇa*s (and the names *la* and *ga* used for any "leftover" *laghu*s and *guru*s respectively), assigning them syllable weights (via vowel lengths) such that the three syllables starting at any of the 8 consonants are itself in the *gaṇa* named by that consonant.[10]

[8] `http://sarit.indology.info`

[9] `http://sanskritlibrary.org`

[10] In the modern terminology of combinatorics, this is a de Bruijn sequence.

Thus we can see that the *gaṇa* names are a useful mnemonic and space-saving device, and yet at the same time, from an information-theoretic point of view, they contain absolutely no information that is not present in the expanded string (the pattern of Ls and Gs). Moreover, for a typical reader who is *not* trying to memorize the definitions of metres (either in the GGLGGLLGLGLG form or the "*ta ta ja ra*"' form), the *gaṇa*s add no value and serve only to further mystify and obscure the topic. Moreover they can be misleading as to the nature of *yati* breaks in the metre, as the metre being described is rarely grouped into threes, except for certain specific metres (popularly used in *stotra*s) such as भुजङ्गप्रयातम्, तोटकम्, and स्रग्विणी. One can as easily (and more enjoyably) learn the pattern of a metre by committing a representative example (a good verse in that metre) to memory, rather than the definition using *gaṇa*s, as the author and others know from personal experience. For these reasons, the *gaṇa* information is de-emphasized in the tool described in this paper.

4.1.2 pādānta-laghu

Sanskrit poetic convention is that the very last syllable in a verse can be *laghu* even if the metre requires it to be *guru*. Consider for instance, the very first verse of Kālidāsa's *Meghadūta*, in the *Mandākrāntā* metre:

> kaścit kāntā-viraha-guruṇā svādhikārāt pramattaḥ
> śāpenāstaṃgamita-mahimā varṣa-bhogyeṇa bhartuḥ
> yakṣaś cakre janaka-tanayā-snāna-puṇyodakeṣu
> snigdhacchāyā-taruṣu vasatiṃ rāmagiryāśrame**ṣu**

Even though the *Mandākrāntā* requires in each *pāda* a final syllable that is *guru*, the final syllable of the verse above is allowed to be *ṣu* which if it occurred in another position (and not followed by a consonant cluster) would be treated as a *laghu* syllable. A similar convention, though not always stated as clearly in texts in prosody, more or less applies at the end of each half (*ardha* or pair of *pāda*s) of the verse (for an example, see the *kāṣṭhād agnir…* verse in *Śālinī* from Section 1.1).

The question of such a *laghu* at the end of *odd pāda*s (*viṣama-pādānta-laghu*) is a thorny one, with no clear answers. Even the word of someone like Kedārabhaṭṭa cannot be taken as final on this matter, as it needs to hold up to actual usage and what is pleasing to the trained ear. Certainly we see such *laghu*s being used liberally in metres like *Śloka*, *Upajāti* and

Vasantatilakā. At the same time, there are metres like *Śālinī* where this would be unusual. The summary from those well-versed in the metrical tradition[11] is that such *laghu*s are best avoided (and are therefore unusual, the works of the master poets) in *yati-prabala* metres, those where the *yati* is prominent. This is why, *Śālinī* with 11 syllables to a *pāda* requires a stricter observance of *guru* at the end of odd *pāda*s than a metre like *Vasantatilakā* with 14. As a general rule of thumb, though, such *viṣama-pādānta-laghu*s can be regarded as incorrect in metres longer than *Vasantatilakā*. It is not clear how a computer could automatically make such subjective decisions, so something like the idea (Melnad, Goyal, and P. M. Scharf 2015) of storing a boolean parameter about which metres allow this option, seems desirable. Still, the question of how that boolean parameter is to be chosen remains open.

4.1.3 Is more data always better?

It seems natural that having data about more metres would lead to better decisions and better results, but in practice some care is needed. A common problem is that when there are too many metres in our database, the likelihood of false positives increases. To see this more clearly, imagine a hypothetical case in which every possible combination of *laghu* and *guru* syllables was given its own name as a metre: in that case, a verse intended to be in the metre *Śārdūlavikrīḍtam*, say, with even a single error, would perfectly match some other named metre, and we would be misled as to the truth. A specific case where this happens easily is when a user inputs a single *pāda* but the system tries to treats it as a full verse. In this case, the quarters of the input, as they are much shorter, are more likely to match some metre accidentally. The solution of returning multiple results (a *list* of results rather than a single result) alleviates this problem (cf. the idea of *list decoding* from computer science).

A related problem is the over-precise naming of metres. We know that *Indravajrā* and *Upendravajrā* differ only in the weight of the first syllable, and that the *Upajāti* metre consists of free alternation between them for the four *pāda*s in a verse, as for this particular metre, the weight of the first syllable does not matter too much. However, there exist theorists of prosody who have, to each of the $2^4 = 16$ possibilities (all the ways of combining *Indravajrā* and *Upendravajrā*), given names like *Māyā*, *Premā*,

[11] Śatāvadhānī R. Ganesh, personal communication

Mālā, *Ṛddhiḥ* and so on (A. Mishra 2007). This is not very useful to a reader, as in such cases, the metre in question is, in essence, really more common than such precise naming would make it seem. Velankar (Velankar 1949) even considers the name *Upajāti* as arising from the "displeasure" of the "methodically inclined prosodist".

Another issue is that data compiled from multiple works on prosody (or sometimes even from the same source) can have inconsistencies. It can happen that the same metre is given different names in different sources (Velankar 1949: p. 59). This is very common with noun endings that mark gender, such as *-ā* versus *-aṃ*, but we also see cases where completely different names are used. It can also happen that the same name is used for entirely different metres (see also the confusion about *Upajāti* mentioned below in Section 4.4). For these reasons, instead of storing each metre as a (name, pattern) pair as mentioned earlier, or as the (better) (name, pattern, bool) triple (Melnad, Goyal, and P. M. Scharf 2015), it seems best to store a (pattern, bool, name, source for name) tuple. I started naively, thinking the name of metres is objective truth, and as a result of this project I realized that names are assigned with some degree of arbitrariness.

Finally a couple more points: (1) There exist metres that end with *laghu* syllables, and the code should be capable of handling them. (2) It is better to keep metrical data as data files, rather than code. This was a mistake made in the initial design of the system. Although it did not deter helpful contributors like Dhaval Patel from contributing code-like definitions for each metre, it is still a hindrance that is best avoided. Keeping data in data files is language-agnostic and would allow it to be used by other tools.

Overall, however, despite these issues, on the whole the situation is not too bad, because it is mostly a small set of metres that is used by most poets. Although the repertoire of Sanskrit metres is vast (Deo 2007), and even the set of *commonly* used metres is larger in Sanskrit than in other languages, nevertheless, as with *rāga*s in music, although names can and have been given to a great many combinations, not every mathematical possibility is an aesthetic possibility.[12]

[12]This remark comes from Śatāvadhānī Ganesh who has pointed this out multiple times.

4.2 Transliteration

It appears that accepting input in various input schemes is one of the features of the tool that users enjoy. Although the differences between various input schemes are mostly superficial and easily learned, it appears that many people have their preferred scheme that they would like to employ wherever possible. These are fortunately easy for computers to handle.

As pointed out elsewhere in detail (P. M. Scharf and Hyman 2011), the set of graphemes or phonemes one might encounter in putatively Sanskrit input is larger than that supported by common systems of transliteration like Harvard-Kyoto or IAST. Characters like chandrabindu and ळ will occur in the input especially with modern poetry or verse from other languages. The system must be capable of doing something reasonable in such cases.

A perhaps unusual choice is that the system does not currently accept input in SLP1, even though SLP1 is used internally. The simple reason is that no one has asked for it, and it does not seem that many people type in SLP1. SLP1 is a great internal format, and can be a good choice for inter-operability between separate tools, but it seems that the average user does not prefer typing `kfzRaH` for कृष्णः. Nevertheless this is a minor point as this input method can easily be added if anyone wants it.

In an earlier paper (Melnad, Goyal, and P. M. Scharf 2015), two of the deficiencies stated about the tool by Mishra (A. Mishra 2007) are that:

1. By supporting only Harvard-Kyoto input, that tool requires special treatment of words with consecutive *a-i* or *a-u* vowels (such as the word "प्रउग"). In this tool, as Devanāgarī input is accepted, such words can be input (besides of course by simply inserting a space).

2. That tool does not support accented input, which (Melnad, Goyal, and P. M. Scharf 2015) do because they accept input in SLP1. In this tool, accented input is in fact accepted if input as Devanāgarī. However, as neither this tool nor the one by (Melnad, Goyal, and P. M. Scharf 2015) supports Vedic metre, this point seems moot: Sanskrit poetry in classical (non-Vedic) metre is not often accompanied by accent markers! In this tool, accent marks in Devanāgarī are accepted but ignored.

4.3 Scansion

As a coding shortcut when the program was first being written, I decided to treat *anusvāra* and *visarga* as consonants too for the purposes of scansion, instead of handling them in a special way. To my surprise, I have not had to revise this and eliminate the shortcut, because in every instance, the result of scansion is the same. I am not aware of any text on prosody treating *anusvāra* and *visarga* as consonants, but their identical treatment is valid from the point of view of Sanskrit prosody. This is a curious insight that the technological constraints (or laziness) have given us!

As mentioned in earlier work (Melnad, Goyal, and P. M. Scharf 2015), in later centuries of the Sanskrit tradition, there evolved an option of considering certain *guru* syllables as *laghu*, as a sort of poetic licence, in certain cases. Specifically, certain consonant clusters, especially those containing *r* like *pr* and *hr*, were allowed to be treated as if they were single consonants, at the start of a word. This rule is stated by Kedārabhaṭṭa too, and seems to be freely used in the Telugu tradition even today. A further trend is to allow this option everywhere, based on how "effortlessly" or "quickly" certain consonant clusters can be pronounced, compared with others. A nuanced understanding of this matter comes from a practising poet and scholar of Sanskrit literature, Śatāvadhānī R. Ganesh:[13] this practice arose from the influence of Prākṛta and Deśya (regional) languages (for instance, it is well-codified as a rule in Kannada and Telugu, under the name of *Śithila-dvitva*). It was also influenced by music; Ganesh cites the treatise **चतुर्दण्डीप्रकाशिका**. His conclusion is that as a conscientious poet, he will follow poets like Kālidāsa, Bhāravi, Māgha, Śrāharṣa and Viśākhadatta in not using this exception when composing Sanskrit, but using it sparingly when composing in languages like Kannada where prior poets have used it freely.

With this understanding,[14] the question arises whether the system needs to encode this exception, especially for dealing with later or modern poetry. This could be done, but as a result of the system's robustness to errors, in practice this turns out to be less necessary. Any single verse is unlikely to exploit this poetic licence in every single *pāda*, so the occasional usage of this

[13] personal communication, but see also corroboration at `https://groups.google.com/d/msg/bvparishat/ya1cGLuhc14/EkIqH9NbgawJ`

[14] See another summary here: `https://github.com/shreevatsa/sanskrit/issues/1#issuecomment-68502605`

exception does not prevent the metre from being detected. The only caveat is that this already counts as an error, so verses that exploit this exception would have a slightly lower robustness to further additional errors.

4.4 Identification

It is not enough for a verse to have the correct scansion (the correct pattern of *laghu* and *guru* syllables), for it to be a perfect specimen of a given metre. There are additional constraints, such as *yati*: because a pause is indicated at each *yati-sthāna* (caesura), a word must not cross such a boundary, although separate lexical components of a compound word (*samāsa*) may. Previously (Melnad, Goyal, and P. M. Scharf 2015), an approach has been suggested of using a text segmentation tool such as the Sanskrit Heritage Reader (Huet 2005; Huet and Goyal 2013) for detecting when such a constraint is violated. This would indeed be ideal, but the tool being described in this paper alleviates the problem by displaying the user's input verse aligned to the metre, with each *yati-sthāna* indicated. Thus, any instance of a word crossing a *yati* boundary will be apparent in the display.

Note that we can provide information on all kinds of *Upajāti*, even if they are not explicitly added to our database, a problem mentioned previously (Melnad, Goyal, and P. M. Scharf 2015). *Upajāti* just means "mixture"; the common *upajāti* of *Indravajrā* and *Upendravajrā*, as a metre, has nothing to do with the *upajāti* of *Vaṃśastha* and *Indravaṃśa* (Velankar 1949). In fact, the latter is sometimes known by the more specific name of *Karambajāti*,[15] among other names. Whenever an Upajāti of two different metres is used and input correctly, each of the two metres will be recognized and shown to the user, because different *pāda*s will match different patterns in our index. So without us doing any special work of adding all the kinds of Upajāti to the data, the user can see in any given instance that their verse contains elements of both metres, and in exactly what way. Of course, adding the "mixed" metre explicitly to the data, would be more informative to the user, if the mixture is a common one.

4.5 Display

Once a metre is identified, for some users, telling the user the name of the metre may be enough. However, if we envision this tool being used by anyone

[15] Śatāvadhān R. Ganesh, personal communication

reading any Sanskrit verse (such as Devadatta from Section 1.3), then for many users, being told the name of the metre (or even the metre's pattern) carries mainly the information that the verse is in *some* metre, but does not substantially improve the reader's enjoyment of the verse. Seeing the verse aligned to the metre, with line breaks introduced in the appropriate places and *yati* locations highlighted, helps a great amount. What would help the most, however, is a further introduction to the metre, along with familiar examples that happen to be in the same metre, and audio recordings of typical ways of reciting the metre.

The tool does this, for popular metres (see Figure 1), drawing on another resource (Ganesh 2013). In these audio recordings made in 2013, Śatāvadhānī R. Ganesh describes several popular metres, with well-chosen examples (most recited from memory and some composed extempore for the sake of the recordings). Some interesting background such as its usage in the tradition—a brief "biography" of the metre—is also added for some metres. Although they were not created for the sake of this tool, it was the same interest in Sanskrit prosody that led both to the creation of this tool and to my request for these recordings. Showing the user's verse accompanied by examples of recitation of other verses in the same metre helps the user read aloud and savour the verse they input.

Incidentally, an introduction to metres via popular examples and accompanying audio recordings is also the approach taken by the book *Chandovallarī* (S. Mishra 1999). The examples chosen are mostly from the *stotra* literature, which are most likely to be familiar to an Indian audience. In this way it can complement the recordings mentioned in the previous paragraph, in which the examples were often chosen for their literary quality or illustrative features.

4.6 Getting feedback from users

The main lesson I learned from building this system was the value of making things accessible to as many users as possible, by removing as many barriers as possible. Write systems that are "liberal" in what they accept, but are nevertheless conservative enough to avoid making errors (Postel's law).

There exist users who may not have much computer-science or programming knowledge, but are nevertheless scholars who are experts in a specific subject. For example, India's tech penetration is low; even many Sanskrit scholars aren't trained or inclined to enter verse in standard

transliteration formats. The very fact that they are visiting your tool and using it means that they constitute a self-selecting sample. It would be a shame not to use their expertise. Their contributions and suggestions can help improve the system. In the case of this tool, the link to GitHub discussion pages, and making it easy with a quick link to report issues encountered during any particular interaction, have generated a lot of improvements, both in terms of usability and correctness. A minor example of a usability improvement is setting things up so that the text area is automatically focused when a user visits the web page—this is trivial to set up, but not something that had occurred as something desirable to do. In this case a user asked for it.

Though user feedback guided many design decisions, gathering and acting on more of the user feedback would lead to further improvements.

5 Conclusions and future work

This paper has described a tool for metre recognition that takes various measures to be useful to users as much as possible. In this section, we list current limitations of the tool and improvements that can be (and are planned to be) made.

In terms of transliteration, though there are many transliteration schemes supported, even the requirement to be in a specific transliteration scheme is too onerous—instead, the tool must let the user type, and in real-time display its understanding of the user's input, while offering convenient input methods (such as a character picker[16]) that do not require prior knowledge of how to produce specific characters. Similarly on the output side, a user's preferred script for reading Sanskrit (which may not be the same as their input script) should be used and remembered for future sessions, so that for instance a user can completely use the tool and see all Sanskrit text in the Kannada script. There may even exist users who prefer to read everything in SLP1!

Very few *mātrā* metres are currently supported (only members of the *Āryā* family have been added). There are many simple *mātrā* metres used in *stotra*s, such a metre consisting of alternating groups of 3 and 4 *mātrā*s. More examples for each metre, such as from *Chandovallarī* (S. Mishra 1999), would help.

[16]For instance, `https://r12a.github.io/pickers/devanagari`

The program is a monolithic application. It should be made more modular, and packaged into libraries for distribution, so that other software can easily incorporate the same user-friendly features. Similarly, in addition to the human interface, providing an API would make this code usable from another website or application. Another limitation is that the program requires a dedicated server to run; if rewritten to run entirely in the browser it could be packaged as a browser extension, so that any Sanskrit verse on any web page can be quickly queried about and reformatted in a metrically clear form. The automatic inference of the transliteration scheme and other aspects of the user's intention, though a user-friendly feature, might have errors occasionally, so the program would be improved by allowing them to be indicated manually when desired.

Finally, the most promising avenue for future work is running this tool on large texts rather than for one verse at a time, which can uncover many insights about prosody. For instance, the most common Anuṣṭubh (Śloka) metre, the work-horse of Sanskrit literature and beloved of the epic poets of the Rāmāyaṇa and the Mahābhārata, is still difficult to define clearly. The naive definition, that the odd *pāda*s match the regular expression "`....LGG.`" and the even *pādā*s match "`....LGL.`", is found insufficient: there are both more and fewer constraints in practice. It is not the case that all 2^{16} choices for the first four syllables are acceptable, nor is it the case that every acceptable *śloka* satisfies even these constraints. G. S. S. Murthy (Murthy 2003) surveys and summarizes the literature on this metre and concludes with some perceptive remarks:

> It is indeed surprising that anuṣṭup has remained ill-defined for so long. […] If anuṣṭup is being used for thousands of years in saṃskṛt literature without a precise definition having been spelt out till date, it must be simply because of the fact that the internal rhythm of anuṣṭup becomes ingrained in the mind of a student of saṃskṛt at an early age due to constant and continuous encounter with anuṣṭup and when one wants to compose a verse in anuṣṭup, one is guided by that rhythm intuitively.

It is now almost within reach, by running a tool like this on a large corpus consisting of the Mahābhārata, Rāmāyaṇa and other large works, to arrive at a descriptive definition of *śloka* based on the verses actually found in the

literature, so that we can make explicit the rules that have been implicitly adhered to by the natural poets.

Acknowledgements

I am indebted to the poet and scholar Śatāvadhānī R. Ganesh for encouraging my interest in Sanskrit (and other Indian) prosody. It is his intimate love of metres (reminding me of the story of the mathematician Ramanujan for whom every positive integer was a personal friend), that led me to the realization that an understanding of metre greatly enriches the joy of poetry. Dhaval Patel contributed metrical data, and raised points about nuances, from *Vṛttaratnākara* (some still unresolved). Sridatta A pointed out some more. I thank Vishvas Vasuki for being a heavy user and pointing out many bugs and suggestions, and for initiating the sanskrit-programmers mailing list where this project began. Finally, I thank my wife Chitra Muthukrishnan for supporting me during this work, both technically and otherwise, and for reviewing drafts of this article.

References

Deo, Ashwini S. 2007. "The metrical organization of Classical Sanskrit verse." *Journal of linguistics* 43.1: 63–114.

Ganesh, Shatavadhani R. 2013. *Sanskrit Metres (A playlist with a series of audio recordings containing recitation and information about popular metres).* URL: `https://www.youtube.com/playlist?list=PLABJEFgj0PWVXr2ERGu2xtoSXrNdBs5xS`.

Huet, Gérard. 2005. "A functional toolkit for morphological and phonological processing: application to a Sanskrit tagger." *Journal of Functional Programming* 15.4: 573–614.

Huet, Gérard and Pawan Goyal. 2013. "Design of a lean interface for Sanskrit corpus annotation." *Proceedings of ICON 2013, the 10th International Conference on NLP*: 177–86.

Melnad, Keshav, Pawan Goyal, and Peter M. Scharf. 2015. "Identification of meter in Sanskrit verse." In: *Selected papers presented at the seminar on Sanskrit syntax and discourse structures, 13–15 June 2013, Universite Paris Diderot, with a bibliography of recent research by Hans Henrich Hock.* Providence: The Sanskrit Library, 325–346.

Mishra, Anand. 2007. *Sanskrit metre recognizer.* URL: `http://sanskrit.sai.uni-heidelberg.de/Chanda/`.

Mishra, Sampadananda. 1999. *Chandovallari: Handbook of Sanskrit prosody.* Sri Aurobindo Society.

Murthy, G. S. S. 2003. "Characterizing Classical Anuṣṭup: A Study in Sanskrit Prosody." *Annals of the Bhandarkar Oriental Research Institute* 84: 101–15. ISSN: 03781143.

—. 2003? *Maatraa5d.java.* URL: `https://github.com/sanskrit-coders/sanskritnlpjava/tree/master/src/main/java/gssmurthy`.

Ollett, Andrew. 2012. "Moraic Feet in Prakrit Metrics: A Constraint-Based Approach." *Transactions of the Philological Society* 110.12: 241–282.

Scharf, Peter. 2016. "Sanskrit Library conventions of digital representation and annotation of texts, lexica, and manuscripts." In: *ICON 2016 Workshop on bridging the gap between Sanskrit computational linguistics tools and management of Sanskrit digital libraries 17–20 December 2016, IIT-BHU.*

Scharf, Peter M. and Malcolm D. Hyman. 2011. *Linguistic Issues in Encoding Sanskrit*. The Sanskrit Library, Providence and Motilal Banarsidass, Delhi. URL: `http://sanskritlibrary.org/Sanskrit/pub/lies_sl.pdf`.

Smith, John. 1998? *sscan (part of sktutils.zip)*. URL: `http://bombay.indology.info/software/programs/index.html`.

Velankar, H. D. 1949. *Jayadāman: A collection of ancient texts on Sanskrit Prosody and A Classified List of Sanskrit Metres with an Alphabetical Index*. Haritoṣamālā, pp. 14–15.