Skrutable: Another Step Toward Effective Sanskrit Meter Identification

Tyler Neill
Leipzig University
Institute for Indology and Central Asian Studies
Schillerstraße 6, 04109
Leipzig, Germany
tyler.g.neill@gmail.com

Abstract

There has been good progress toward making computational identification of Sanskrit meter accurate and easy to use. The present project continues this work with a new online tool offering whole-file analysis and detailed scansion output. It also presents a preliminary meter analysis data set of 150,000 verses from a range of genres as a resource to be collaboratively developed to help power future, machine learning-based iterations of the identification task.

1 Introduction

It is already very clear that verse is an important part of Sanskrit literature, and also that having a computer tool to help identify the meter of a given verse is useful. Previous work has laid out the numerous motivations for computational work in Sanskrit meter in particular (Ingalls, 1985; Smith, 1999; Murthy, 2003; Sellmer, 2013), explained the theoretical background (Deo, 2007; Rama N. and Lakshmanan, 2010; Hahn, 2014; Melnad et al., 2015; Rajagopalan, 2018; Scharf, 2018), and presented several usable tools already (especially those of Melnad et al. 2015 and Rajagopalan 2018).

Taken together, this represents good progress toward making computational identification of Sanskrit meter accurate and easy to use. The present project continues this work with a new online tool offering whole-file analysis, detailed scansion output, and a number of other features. It also presents a preliminary meter analysis data set of 150,000 verses from a range of genres as a resource to be collaboratively developed to help power future, machine learning-based iterations of the identification task.

1.1 Basic Task

The basic meter identification task is as follows:

> Given a Sanskrit verse as textual input, output information useful for assessing how it compares to known metrical patterns, including first and foremost a traditional label when possible (e.g., "anuṣṭubh", "śārdūlavikrīḍitā", "āryā", etc.) but also further scansion details for inspection especially in the case of greater variation from conventional patterns.

1.2 Standard Approach

The standard approach to this task reproduces the mechanical method of the student, especially one not immersed in an oral tradition:

1. Understand the textual input as Sanskrit (the data cleaning and "transliteration" step).
2. Break the text into syllables using one simple set of rules, and evaluate each syllable as either light or heavy using another set of rules (the "scansion" step).

3. Compare this pattern of syllable weights against a set of known patterns, looking for a best match (the "identification proper" step).

Depending on particular interests and goals, more detail is also possible, including other analytic units (e.g., pādas, gaṇas, yatis, etc.), specific treatment of partial matches, additional output features (e.g., traditional definitions, sound files), and so on.

1.3 Specific Motivation

Because most users of Sanskrit are not (yet) also computer programmers, a tool that can perform this task must also be user-friendly enough that, for example, a philologist can easily analyze one or more entire texts in this way without needing to know what a "for-loop" is. If students and scholars can be empowered to interact with large amounts of textual data in this way, then not only does their individual research stand to gain, but also, if they aggregate their results, everyone can benefit. Namely, not only do these results immediately serve for scholarly reference, but also, if designed to be machine-readable, they can additionally serve as training data for machine learning-based upgrades of the identification task. One possible such upgrade could consist in the detection of metrical material from among non-metrical material using sequence-to-sequence labeling.

2 Skrutable

The current project, called Skrutable (as in "let's make Sanskrit less inscrutable"), provides just such a forward-looking resource. It may not outperform its peers in every last technical respect (see §4), but it does certainly contribute by empowering its users with access to whole-file analysis and a number of other engaging features. Furthermore, its modular design will allow it to be improved through user feedback and open-source cross-pollination in due time.

2.1 Relation to Previous Work

The present project began in 2015 with direct inspiration from the "Meter Identifying Tool" (at `http://www.sanskritlibrary.org:8080/MeterIdentification`, hereafter "MIT-2015") of Keshav Melnad, Pawan Goyal, and Peter Scharf (Melnad et al., 2015). Two years earlier, in 2013, I had also met and spoken with Anand Mishra in Heidelberg, whose similar 2007 "Sanskrit metre recognizer" tool (formerly at `http://www.sanskrit.sai.uni-heidelberg.de/Chanda/HTML`) is today unfortunately no longer found online. Further motivation to continue developing a Python-based toolkit for Sanskrit came from speaking with two creators of the Classical Language Toolkit (Kyle Johnson and Patrick Burns) at a conference in Leipzig in 2017. Then, while late in the project, I found out about Shreevatsa Rajagopalan's "Metre Identifier" (at `http://www.sanskritmetres.appspot.com`, hereafter "MI-2018"), begun in 2013 and presented in 2018. Finally, while revising this paper, I became aware that work on MIT-2015 had continued (Scharf, 2018).[1] In accordance with my interest in what is easily accessible to

[1] This 2018 paper by Scharf introduced TEITagger, a system for preparing digital editions of texts in accordance with the Text-Encoding Initiative Guidelines. Sections §4–5 of this paper ("TEITagger software" and "Philological use of the TEITagger software", respectively) discuss the further development of MIT-2015, its incorporation into TEITagger, and its application to the Mahābhārata. The paper also does a good job of discussing the following additional previous scholarship, the citations for which are worth mentioning in full:

> Edgerton, Franklin. 1939. "The epic triṣṭubh and its hypermetric varieties." Journal of the American Oriental Society 59.2: 159–74. doi: www.jstor. org/stable/594060.
> Fitzgerald, James L. 2004. "A meter-guided analysis and discussion of the dicing match of the Sabhāparvan of the Mahābhārata."
> —. 2006. "Toward a database of the non-anuṣṭubh verses of the Mahābhārata." In: Epics, Khilas, and Purāṇas: continuities and ruptures. Proceedings of the Third Dubrovnik International Conference on the Sanskrit Epics and Purāṇas. Ed. by Petteri Koskikallio. Zagreb: Croatian Academy of Sciences and Arts, pp. 137–48.
> —. 2009. "A preliminary study of the 681 triṣṭubh passages of of the Mahābhārata." In: Epic undertakings: proceedings of the 12th World Sanskrit Conference. Ed. by Robert Goldman and Muneo Tokunaga. Delhi: Motilal Banarsidass, pp. 95–117.

all, I will focus my comparative discussion here only on the two tools that are freely available online: MIT-2015 and MI-2018.

For step 1 (data cleaning and transliteration, see §1.2 above), each of the Sanskrit meter projects named above appears to use its own code for transliteration, drawing on neither the "Sanscript" package (`https://www.github.com/sanskrit/sanscript`) — (I surmise) originally written collaboratively in Javascript and also PHP by the online "Sanskrit Coders" / "Sanskrit Programmers" group but adapted for easy importation in Python in 2013 by Arun K. Prasad — nor the "Aksharamukha" (`https://www.github.com/virtualvinodh/aksharamukha`) by Vinodh Rajan, going back to 2010 and also importable in Python. As a result, there is varying support for transliteration schemes among the different meter projects, and additional use of other tools is sometimes required. For its part, Skrutable, written in Python 3, currently supports the following: Unicode-based International Alphabet for Sanskrit Transliteration (IAST); ASCII-based "Sanskrit Library Phonetic Basic" (SLP1),[2] Harvard-Kyoto (HK), Velthuis (VH), Indian languages Transliteration (ITRANS), "WX" (scheme developed at IIT Kanpur); and Unicode-based Devanagari, Gujarati, and Bengali.[3]

For step 2 (scansion), each of the three online meter tools (MIT-2015, MI-2018, and Skrutable) also solves the relatively simple problems in syllabification and weight calculation on their own, to no major effect on subsequent processing, as far as I can tell. Instead, their main differences lie in the precise handling of step 3, the "identification proper" step (see §1.2 above) and in what other features they offer (e.g., display of scansion information).

2.2 Algorithm

Skrutable's basic algorithm for meter identification does not differ much from other rule-based programming solutions. At its most basic, once relevant characters have been filtered and transformed (via internal processing in SLP1) into the corresponding sequence of light and heavy syllable weight values, this sequence is run through what I will call a "cascade" of checks against regular expressions for known and named patterns. The order of this cascade is fixed, as follows:

- Anuṣṭubh śloka
  - two perfect halves
  - two halves, one perfect, one imperfect
  - a single perfect half
- Varṇavṛttas
  - perfect samavṛtta
  - perfect ardhasamavṛtta
  - perfect viṣamavṛtta
  - perfect upajāti (triṣṭubh etc.)
  - imperfect samavṛtta (two or three out of four pādas)
  - imperfect upajāti (two or three out of four pādas)
- Mātrāvṛttas a.k.a. jātis (perfect only)

If ever an overall perfect match (four out of four perfect pādas) is found, the cascade breaks and the match is reported. All verses are treated as consisting of four parts, and identification of overall imperfect verses currently relies on the internal support of a certain number of perfect individual components within the same verse. As a consequence of this design, not currently supported are: 1) verse types with numbers of pādas other than four (e.g. the three-pāda

[2] More accurately, this project uses only a subset of SLP1, e.g. being less strict about punctuation.

[3] Character frequency-based automatic detection of schemes is also soon to be supported but is still under repair at the time of this publication.

gāyatrī and five-pāda paṅkti of Vedic literature); 2) imperfect ardhasamavṛtta, viṣamavṛtta, or jāti verses; and 3) all cases of partial verses (e.g., single pādas and half verses) or overfull verses (i.e., anything over four pādas) except half-anuṣṭubh. Also, relatively few samavṛttas (less than 100) have been programmed for recognition.

This cascade is then assisted by pāda-resplitting, which happens outside of it. That is, Skrutable always first uses its cascade to try to identify the verse with pāda breaks as given by the user. Having this no-resplit, "WYSIWYG" behavior at the core of the identification algorithm was seen as desirable for both practical and pedagogical purposes. As soon as this fails, however, Skrutable can in turn also produce a series of alternative pāda-break configurations (hereafter: "resplits") for similar cascade testing. This resplitting is initialized by accepting user input as provided (supported string sequences are set as a config variable) and otherwise by dividing the overall length in syllables into four. Resplit points are then moved progressively further out from their starting points, which privileges more likely configurations earlier and helps the identification process successfully complete sooner. Experience found use for two additional parameters here: 1) an option to further privilege the frequently informative user-input midpoint of the verse, i.e., the "bc" pāda break (hereafter: the "keep mid" option); and 2) a distance option determining the total maximum number of resplits to be attempted, with one setting ("resplit max") looking further and one option ("resplit lite") quitting sooner. Skrutable's approach to pāda breaks is thus similar to that of MI-2018 in its concern to reduce the number of resplits tested,[4] but different in its implementation.

The third major component of Skrutable's identification algorithm is result scoring. For as long as no perfect match is found, resplitting and testing against the cascade continues. All results, including partial matches defined by two or three correct pādas out of four (for supported verse types), are collected and scored according to an empirically calibrated look-up table (values being again set as config variables). The highest scoring result is reported as the best match, and other results (as well as statistics on total resplits i.e. identification attempts) are stored as an internal object attribute. In the rare case of a tie as determined by the scoring system, both results are reported with the phrase "atha vā" ("or").

In this way, although resplitting greatly increases the number of checks performed, it allows for comprehensive handling of even difficult cases, even as input in a natural way, and it leads to better performance in cases with pādas of uneven length, either inherently (e.g., jāti, ardhasamavṛtta) or because of data errors. Also, on a modern machine, such as the server currently running the public-facing deployment of the tool, total computation time per verse generally remains in the millisecond range (cp. §4.3 below).

2.3 Implementation

Code

Skrutable is written as an importable Python 3 library with modules for transliteration, scansion, and meter identification.[5] In combination with an associated front-end user interface (hereafter: "the UI") written primarily in Flask, and secondarily with HTML, CSS, and Javascript, Skrutable also functions as a web app, which is currently deployed online at `https://www.skrutable.info` and which can also be installed and run locally if desired.[6] All code and content is on GitHub,[7] and documentation is provided throughout, for both developer and student consultation.

[4] Cp. Rajagopalan 2018, pp. 125ff.

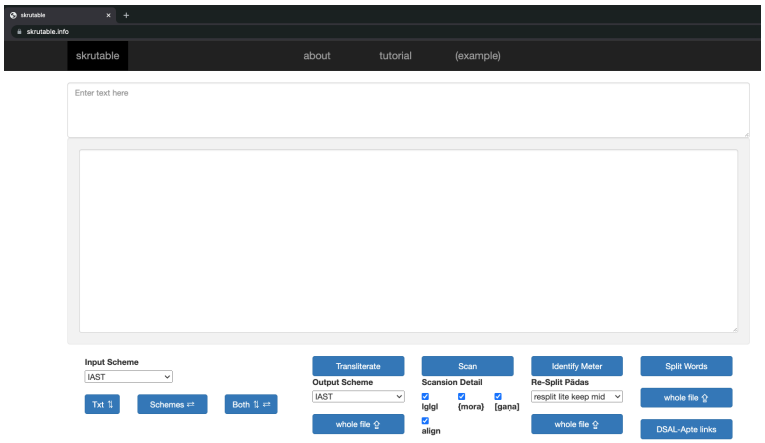
[5] A simple wrapper module also provides access to a pretrained version of the Hellwig-Nehrdich Sanskrit Sandhi and Compound Splitter (Hellwig and Nehrdich, 2018).

[6] The web interface was initially found at the address `http://www.skrutable.pythonanywhere.com`.

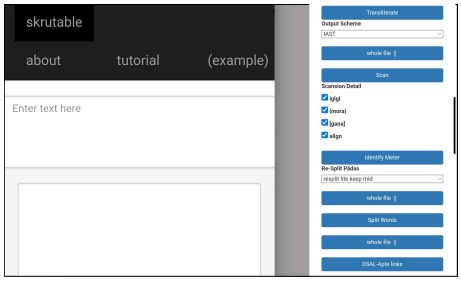
[7] Library at `https://www.github.com/tylergneill/skrutable`, and front-end at `https://www.github.com/tylergneill/skrutable_front_end`.

User Interface

The Skrutable UI (Fig. 1) is designed to provide convenient access to the library's three major functions and to help tailor output to user needs.



(a) On desktop device



(b) On mobile device

Figure 1: Screenshots of blank UI

These three major functions and their interrelations are represented in the UI by its centermost top blue buttons, read left to right (see Fig. 2). That is, after first controlling for the input text's transliteration scheme, Skrutable can next "scan" the input text for its straightforward and essential metrical properties. Then, if desired, it can also, in a third and final separate step, attempt to "identify" the verse as being of a particular meter. Showing the user two separate buttons for the latter two functions is meant to help clarify this important difference.

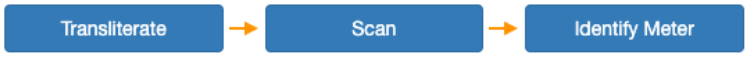


Figure 2: *Major function buttons*

As explained above (§2.2), when the "Scan" button is clicked, Skrutable's "scansion" module does not remove pāda breaks, such that this basic stage of the output exactly reflects the breaks in the input.[8] Scansion output detail (hereafter just "scansion detail") is broken down into four optional components, each of which can be toggled with its own checkbox below the "Scan" button: 1) raw syllable weights, 2) mora count, 3) syllable count and grouping into gaṇa abbreviations, and 4) syllable-to-weight alignment (using a simple monospace font with fixed per-character width plus spaces for filler). In general, any Romanization scheme would work well for this scansion output, but since Indic scripts do not yet also display well in this context, I opt to output only IAST for now.

Then, if "none" is chosen as the meter identification "Re-Split Pādas" option using that drop-down box and if the "Identify Meter" button is clicked, Skrutable's "meter_identification" module first internally invokes the separate "scansion" module to fetch the appropriate scansion information, and then it runs the identification cascade a single time with the pāda breaks as given and outputs the result. If the verse was, for example, not input as four separate lines, then this no-split identification will certainly fail, regardless of the textual content (see Fig. 3). In fact, I see this possibility for very literal behavior as desirable from a pedagogical standpoint. That is, a user may, if he or she wishes, use the scansion output to iteratively work toward a proper pāda break solution him- or herself, visually scrutinizing patterns, manually changing

[8] Surrounding or multiple pāda-break sequences are currently treated as extraneous and simplified.

the input pāda breaks, adding or subtracting "scansion detail" features, and again clicking the "Scan" and/or "Identify Meter" buttons, until a positive result is found.

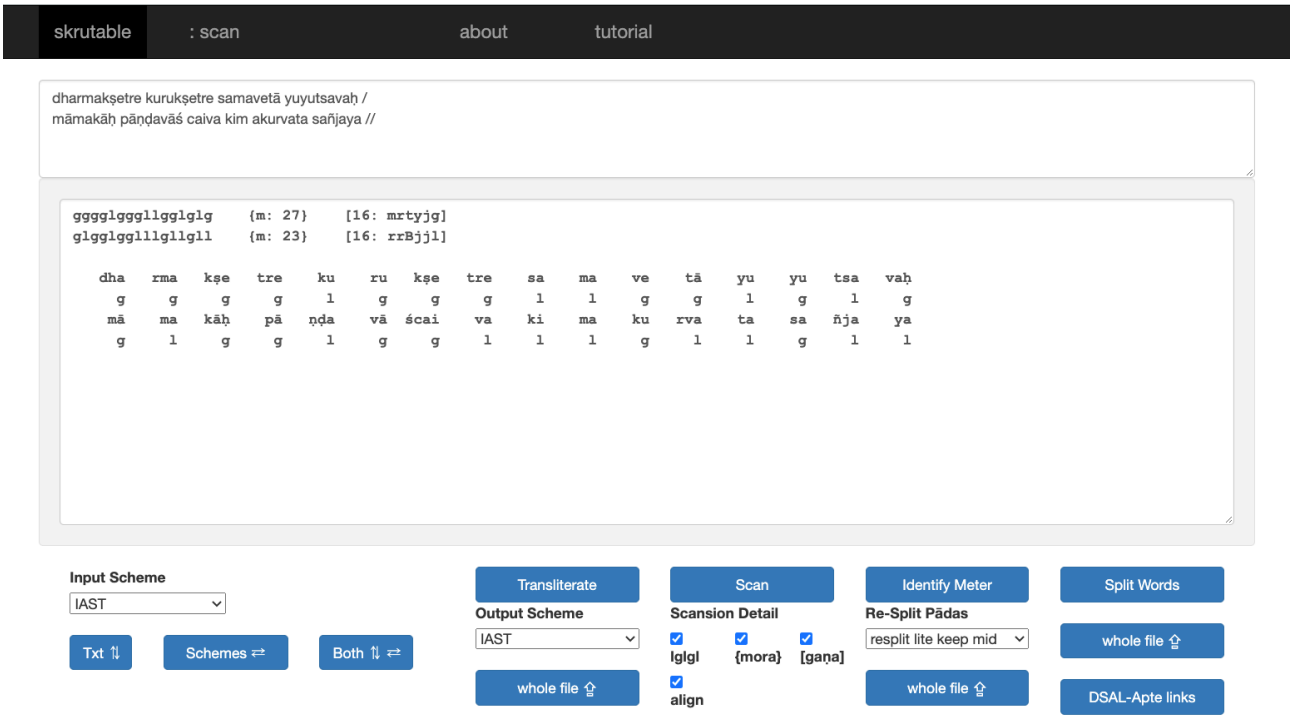


Figure 3: *Meter id., resplit "none", on two lines, with failure id. label "na kiṃcid adhyavasitam"*

A few more things to note here: 1) Both input and output are in plain-text, making them easy to play with and/or copy-paste as needed. 2) Swap buttons are also provided for text buffers and input schemes. 3) The "scansion detail" options apply equally for both the "Scan" and "Identify Meter" buttons. 4) Text-buffer contents and all selected options will persist between visits to the site.[9]

Frequently, however, a user will not be interested in splitting the verse manually in this way, and so the meter identification module also includes functionality for resplitting, described above (§2.2). Internally, scansion and the identification cascade are simply repeated for each resplit, in a simple loop. In the UI, however, when "Identify Meter" is clicked with a resplit option active, all that appears as output is the highest-ranking pāda break configuration, its scansion details, and a label describing the result (Fig. 4).

For users also interested in multi-verse, i.e., whole-file metrical analysis, it is then as easy as looking past the "Identify Meter" button and instead clicking the "whole file" upload button located below it. One then can follow the instructions to upload a file,[10] and once processing and downloading are finished, output can be retrieved from one's normal browser download location. This output consists of a plain-text file (Fig. 5), containing for each verse its full text, its scansion detail according to chosen settings, and its meter identification label. Additionally, if desired, the system can also be made to output a summary of detected verse types in tab-separated value (TSV) format (Fig. 6).[11] In this table, perfect meter types and their associated frequencies appear to the left, while imperfect outcomes and their frequencies appear to the right.

[9]This out-of-the-box, client-side "*session*" feature of the Flask web-development framework works essentially like a cookie.

[10]The file currently must be in plain-text, properly encoded in UTF-8 and in the chosen transliteration scheme, with one verse per line, and with an ASCII-only filename.

[11]This recently developed feature is demonstrated in the "Jupyter Notebook" folder of the GitHub repo and should be incorporated into the UI before long.

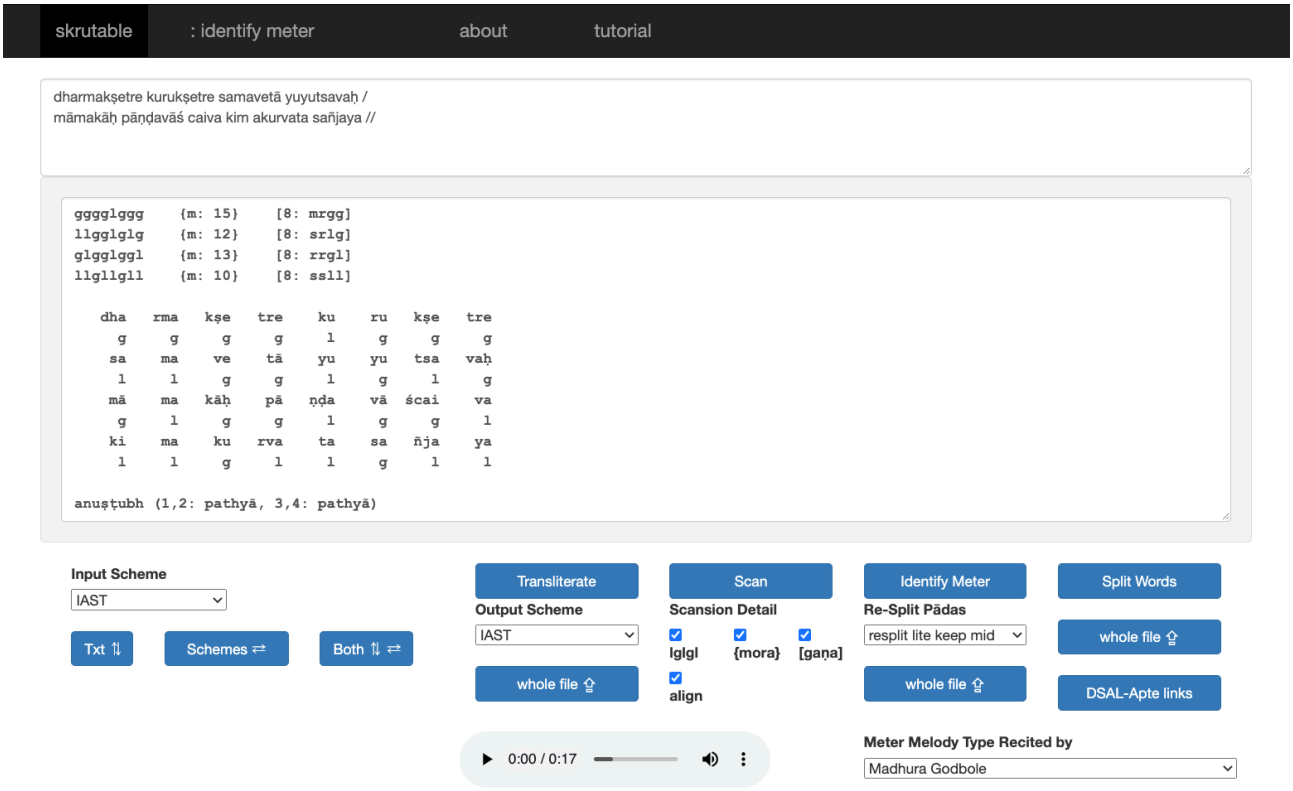


Figure 4: *Meter id., "resplit lite keep mid", with correct four-pāda solution (pathyā anuṣṭubh)*

Whole-file processing takes just a few seconds in most cases (see §4.3 below), so trial-and-error adjustment of input content and settings is not costly.

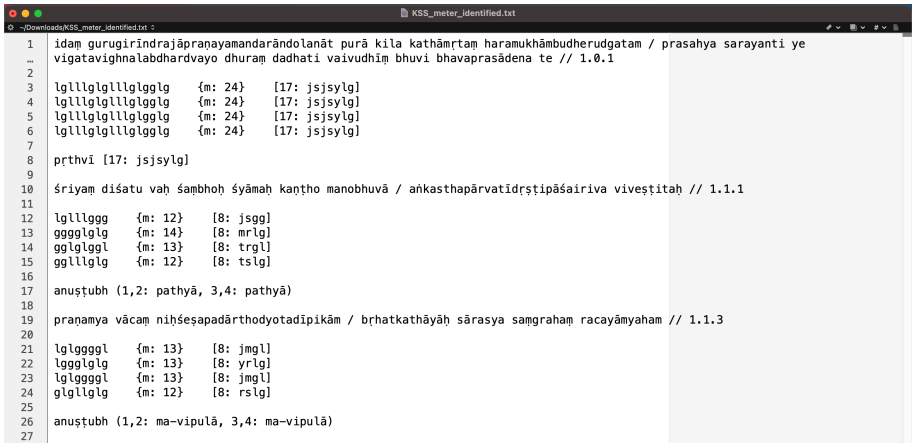
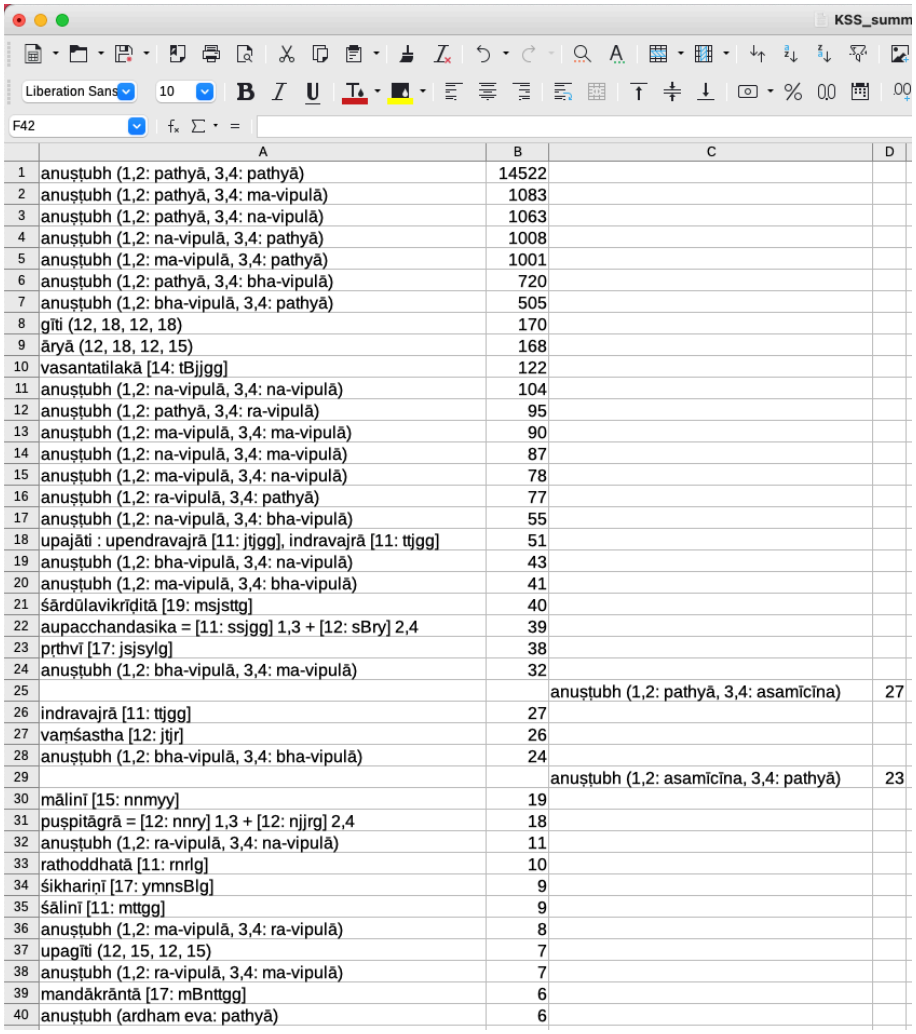


Figure 5: *Text-file output (in BBEdit), KSS examples (pṛthvī, anuṣṭubh pathyā, ma-vipulā)*

Finally, in addition to the meter identification functionality already described, the UI also offers a few more power features, both regarding meter and otherwise:[12]

- Whole-file transliteration
- Transliteration scheme encoding standardization (e.g., to ensure precomposed IAST "ṝ")
- A melody player for successful single-verse meter id. (multiple voice/melody options, details available at `https://www.skrutable.info/reciters`)

[12] Additional features still undergoing development in late 2021 include: Indic-style (i.e. "varṇamālā"/"akṣaramālā"-order) list alphabetization; virāma options for Indic output (both more and less); and options for punctuation preservation in word splitting.



	A	B	C	D
1	anuṣṭubh (1,2: pathyā, 3,4: pathyā)	14522		
2	anuṣṭubh (1,2: pathyā, 3,4: ma-vipulā)	1083		
3	anuṣṭubh (1,2: pathyā, 3,4: na-vipulā)	1063		
4	anuṣṭubh (1,2: na-vipulā, 3,4: pathyā)	1008		
5	anuṣṭubh (1,2: ma-vipulā, 3,4: pathyā)	1001		
6	anuṣṭubh (1,2: pathyā, 3,4: bha-vipulā)	720		
7	anuṣṭubh (1,2: bha-vipulā, 3,4: pathyā)	505		
8	gīti (12, 18, 12, 18)	170		
9	āryā (12, 18, 12, 15)	168		
10	vasantatilakā [14: tBjjgg]	122		
11	anuṣṭubh (1,2: na-vipulā, 3,4: na-vipulā)	104		
12	anuṣṭubh (1,2: pathyā, 3,4: ra-vipulā)	95		
13	anuṣṭubh (1,2: ma-vipulā, 3,4: ma-vipulā)	90		
14	anuṣṭubh (1,2: na-vipulā, 3,4: ma-vipulā)	87		
15	anuṣṭubh (1,2: ma-vipulā, 3,4: na-vipulā)	78		
16	anuṣṭubh (1,2: ra-vipulā, 3,4: pathyā)	77		
17	anuṣṭubh (1,2: na-vipulā, 3,4: bha-vipulā)	55		
18	upajāti : upendravajrā [11: jtjgg], indravajrā [11: ttjgg]	51		
19	anuṣṭubh (1,2: bha-vipulā, 3,4: na-vipulā)	43		
20	anuṣṭubh (1,2: ma-vipulā, 3,4: bha-vipulā)	41		
21	śārdūlavikrīḍitā [19: msjsttg]	40		
22	aupacchandasika = [11: ssjgg] 1,3 + [12: sBry] 2,4	39		
23	pṛthvī [17: jsjsylg]	38		
24	anuṣṭubh (1,2: bha-vipulā, 3,4: ma-vipulā)	32		
25			anuṣṭubh (1,2: pathyā, 3,4: asamīcīna)	27
26	indravajrā [11: ttjgg]	27		
27	vaṃśastha [12: jtjr]	26		
28	anuṣṭubh (1,2: bha-vipulā, 3,4: bha-vipulā)	24		
29			anuṣṭubh (1,2: asamīcīna, 3,4: pathyā)	23
30	mālinī [15: nnmyy]	19		
31	puṣpitāgrā = [12: nnry] 1,3 + [12: njjrg] 2,4	18		
32	anuṣṭubh (1,2: ra-vipulā, 3,4: na-vipulā)	11		
33	rathoddhatā [11: rnrlg]	10		
34	śikhariṇī [17: ymnsBlg]	9		
35	śālinī [11: mttgg]	9		
36	anuṣṭubh (1,2: ma-vipulā, 3,4: ra-vipulā)	8		
37	upagīti (12, 15, 12, 15)	7		
38	anuṣṭubh (1,2: ra-vipulā, 3,4: ma-vipulā)	7		
39	mandākrāntā [17: mBnttgg]	6		
40	anuṣṭubh (ardham eva: pathyā)	6		

Figure 6: *TSV-file output (in LibreOffice), KSS, top 40 meters, with frequencies*

- Point-and-click access to the powerful plain-text Sanskrit Sandhi and Compound Splitter[13]
- Conversion of split words into hyperlinks for V.S. Apte's Practical Sanskrit-English Dictionary (1957-59 ed.) as provided by Digital Dictionaries of South Asia (allied with Digital South Asia Library or DSAL)

3 Data

Here I will describe the textual data, in total about 150,000 Sanskrit (and Prakrit) verses, pertaining to the assessment of Skrutable's meter identification system.

3.1 Input

Data was sourced largely from GRETIL. Other sources used include 1) the author's own repository of NLP-ready philosophical texts,[14] and 2) the repository of Vishvas Vasuki.[15]

[13] Using the pretrained model at `https://www.github.com/OliverHellwig/sanskrit/tree/master/papers/2018emnlp` (Hellwig and Nehrdich, 2018). The Skrutable wrapper also supports unlimited input length and punctuation preservation.

[14] At `https://www.github.com/tylergneill/pramana-nlp`, used here specifically for Ślokavārtika, Pramāṇavārtika, Pramāṇavārtikālaṃkāra, and Tattvasaṃgraha. In addition to GRETIL, this repository uses data from 1) SARIT, 2) the unofficially circulating work of Jongcheol Lee and Helmut Krasser, and 3) original digitization.

[15] At `https://www.github.com/sanskrit/raw_etexts`, used here specifically only for Mālatīmādhava. This repository draws on `http://www.sanskrit.nic.in/ebooks.php`, a number of other scraped text repositories, and also some unknown sources.

The following nineteen texts from various genres (Epic, Story, Kāvya, Philosophy) and generally of large size were chosen for analysis: Mahābhārata (MBh), Kathāsaritsāgara (KSS), Rāmāyaṇa (Ram), Pañcatantra (PT_vv), Mahāsubhāṣitasaṃgraha (MSS), Śiśupālavadha (SpV), Raghuvaṃśa (RVams), Kirātārjunīya (Kir), Sattasaī/Saptaśatī (SS, in Prakrit), Kumārasaṃbhava (KuS), Meghadūta (MgD), Śatakatraya (ST), Mālatīmādhava (MM_vv), Tattvasaṃgraha (TS), Ślokavārtika (SV), Pramāṇavārtikālaṃkāra (PVA_kk), Pramāṇavārtika (PV_kk), Bodhicāryāvatāra (BCA), and Mūlamadhyamakakārikā (MMK). Metrical details for the individual texts, including their respective lengths in verses, are given below in Figs. 7–10.

Original files are provided in the Skrutable_front_end GitHub repo, along with cleaned versions actually used for input. Cleaning interventions included 1) extracting verse from "mixed"-style (i.e., miśraka) texts (PV1_kk from PV-Svavṛtti, TS from TS-Pañjikā, PVA_kk, and MM_vv); 2) re-applying sandhi in certain of these cases (PV_kk); and 3) adjusting numbering, for example, breaking verses that obviously consist of three or more verse halves (pādadvayas) into multiple verses, or simply removing non-canonical material (e.g., from the digitized critical MBh). Digitization quality (not to mention edition quality) is known to be mixed, but as has been observed (Rajagopalan 2018: 128–130), partly erroneous data is actually a virtue in the present context, as the ability to gracefully handle and illuminate data problems on a large scale is one of most useful potential applications of such tools.

3.2 Output

For the present study, the meter identification settings "resplit max" and "keep mid" were used. The exact output text-files produced by Skrutable, both with all available scansion details and in a more abbreviated form (verse number and meter label only), are available on the GitHub repo. Also provided are TSV summary files generated during the analysis, both for individual texts and overall. It is also easy enough to repeat the analysis for oneself by uploading the same input data to the Skrutable website, for example to see how quickly the script runs on a given text or to try out other settings. Since exact output will also vary depending on ongoing project upgrades, the exact code used for generating this particular data is archived as a package release on Zenodo.[16]

4 Evaluation

For a meter-identification tool like Skrutable, it would be desirable to evaluate four things: 1) identification accuracy, 2) usefulness of output information, 3) speed, and 4) ease of use.

4.1 Accuracy

I'll begin my discussion of system accuracy by providing some background on how the two similar projects up for comparison here have been assessed. MIT-2015 was tested on "a database of 1031 verses in the Pañcākhyānaka, including 291 verses in 23 different types of meters besides Anuṣṭubh" (Melnad et al. 2015: 325). The tool was iteratively calibrated during testing, and overall performance was evaluated as follows: "MIT correctly recognized 1018 out of the 1031 verses (98.7%) and 287 out of the 291 non-Anuṣṭubh verses (98.6%). There were no cases in which a meter was recognized incorrectly" (ibid.: 342). The MIT-2015 software is not easily available online, but I was able to obtain from Peter Scharf updated software for the 2018 TEITagger project (JAR file, v28) into which MIT-2015 was incorporated. I also received the relevant Pañcākhyānaka database file (XML), and with some help, I was able to get TEITagger working on this file to identify the meters. Since these results sometimes differ from those of MIT-2015 online, I cannot yet speak to the above accuracy claims. Time permitting, I will continue to work on this verification process.

In turn, Rajagopalan 2018, while noting that "the lack of a reliable (and nontrivial) 'gold standard' hinders attaching a numeric value to the accuracy" describes how for its tool "a large

[16] Available for download at `https://zenodo.org/badge/latestdoi/103173253` (Skrutable) and `https://zenodo.org/badge/latestdoi/328881004` (Skrutable_front_end).

number of texts such as from GRETIL were examined", again enabling tool calibration and resulting in a "recall of 100% in the examples tested" and a precision that is "lower and harder to measure because of errors in the input" (2018: 128–129). Although the underlying GRETIL texts are themselves relatively accessible, the exact "example" input used is not easily available for inspection, nor is exact output. However, a useful table of summary statistics, not featured in Rajagopalan 2018 but mostly coinciding with the texts mentioned therein, is found on the project website.[17] The data from this table is incorporated into Figs. 7 and 8 below. As for the software itself, I found the Python code relatively easy to download and apply to numerous verses.

In the absence of a gold-standard, Skrutable is not yet evaluated with a percentage accuracy. Instead, in order to enable more direct comparison with other projects and thereby begin to approach such a standard, Skrutable was tested on texts similar to those used to test the other two projects discussed here, and all results are being made publicly available: one text for comparison with MIT-2015, the Pañcākhyānaka a.k.a. Pañcatantra (only the GRETIL version so far), and six texts for comparison with MI-2018, namely, MSS, SpV, RVams, Kir, KuS, and ST (again all GRETIL). In addition to these (seven texts in about 17,000 verses), Skrutable was also tested on considerably more and diverse material (eleven additional texts in about 133,000 verses) in the hopes of attracting the interest of more scholars who can help produce the needed gold-standard annotations. The overall summary statistics are presented here in two pairs of tables, the first pair (Figs. 7 and 8) for material overlapping with previous projects, and the second pair (Figs. 9 and 10) for original analysis of novel data.[18]

Visual inspection of the first pair of tables (Figs. 7 and 8) reveals a great deal of numerical agreement between MI-2018 and Skrutable, but many individual discrepancies as well. The latter are strongly suspected to be due mostly to differences in how imperfect verses are handled (see §4.2 below). For Skrutable, in line with the detailed data it was designed to produce, three additional types of information are given: 1) a few important meter subtypes (for anuṣṭubh and upajāti); 2) two numbers instead of one in those cases (anuṣṭubh, upajāti, and samavṛtta) where Skrutable is made to explicitly distinguish imperfect verses (e.g., for indravajrā in Kirātārjunīya, 18 perfect and 2 imperfect); and 3) explicit labels for specific kinds of categorical failures ("ajñātasamavṛtta", "ajñātārdhasamavṛtta", "no id", and in fact most cases of "non-triṣṭubh upajāti"). In short, Skrutable, like the other tools, seems generally capable of correct identification in the case of good data.

4.2 Usefulness of Output Information

On the other hand, especially in the case of worse data, Skrutable's basic scansion output (cp. Fig. 4) is designed especially for iterative exploration after identification fails. Whereas the user interfaces of both MIT-2015 and MI-2018 use HTML text formatting (color and bold weighting, respectively) to highlight syllable weight patterns, Skrutable's strictly plain-text output — with

[17] `http://www.sanskritmetres.appspot.com/statistics`, sourced from GitHub repo "templates" folder.

[18] Note that Scharf 2018 also presents similar analysis for over 73,000 verses of the Mahābhārata and compares results with Fitzgerald 2009 (see footnote 1 above). I quote here a summary of Scharf's MBh results (p. 242) in full:

> TEITagger version 2 tagged 73,436 verses and 1,057 prose sentences in 386 paragraphs. The verses include 68,860 Anuṣṭubhs, 2,970 Triṣṭubhs, 431 Jagatī, 322 Indravajrā, 0 Upendravajrā, 496 of the standard Upajāti variety alternating the two preceding, 88 Śālā, 78 Vāṇī (other Upajātis), 31 Aparavaktra (an ardhasamavrtta meter), 22 Praharṣiṇī, 16 Rucirā, 9 Mālinī, 4 Vasantatilakā, 4 Puṣpitāgrā, 1 Śārdūlavikrīḍita, 1 Halamukhī, 1 Āryāgīti (a type of Āryā), 1 mixture of half Kāmakrīḍā and half Kāmukī, and a hundred unidentified. The unidentified metrical patterns include for instance, 1 mixture of half Kāmukī and half unidentified, 1 mixture of a deviant pāda with subsequent Anuṣṭubh, jagatī, and Triṣṭubh pādas, as well as 98 other uninvestigated unidentified patterns.

Comparison of this paragraph (or Table 1 in Scharf 2018, p. 244 — or Table 3, pp. 245–6 presenting the results of "TEITagger version 17") with Figs. 9–10 here reveals only a rough correspondence between the two attempts. Clearly a more careful comparison is a desideratum for future study.

(Skr vs. MI and MIT)	Kir		KuS		MSS		RVams		SpV		ST		PT_vv
	1,040		613		9,979		1,637		1,645		324		1,105
anuṣṭubh śloka	(MI)	(Skr)	(etc.)										(MIT) (Skr)
pathyā (MI: all)	125	103, 2	157	120, 1	4457	3309, 185	565	433, 55	231	104, 36	35	29, 1	740 639, 154
vipulā		20		36		629, 14		71, 5		76, 13		5	57, 5
upajāti													
comm.triṣṭ. (MI: all)	132	129	187	180	490	436	463	425	62	37	14	7	21
other triṣṭubh		6		4		63		58		34		2	6
non-triṣṭubh						18, 51		0, 1		18, 2		2	3, 2
samavṛtta													
pramāṇikā					16								
indravajrā	20	18, 2	30	30, 3	149	104, 8	104	66, 24	21	6, 22	1	1	6, 5
upendravajrā	24	22, 1	9	9, 1	41	27, 5	28	9, 16	25	5, 18		0, 1	2
rathoddhatā	38	37, 1	90	88	99	86, 3	151	120, 18	87	80, 3	3	2, 1	1
śālinī	3	3			45	27, 1	1	1	81	63, 5	3	3	1, 1
svāgatā	77	71			102	79, 1	1		91	44, 14			
toṭaka					19	18	1	1	2	1			
drutavilambita	33	32, 1			106	101, 5	64	48, 16	69	37, 28	5	3, 1	2
pramitākṣarā	51	49, 2			35	33, 1			82	52, 23			1
vaṃśastha	214	203, 11	84	83, 1	197	167, 16	74	63, 10	145	40, 34	3	2, 1	7, 5
praharṣiṇī	50	46, 4			42	36, 2	8	6, 2	76	35, 35			
mañjubhāṣiṇī					15	15	1	1	69	59, 10			
rucirā					9	7, 2	1	0, 1	68	62, 6			
vasantatilakā	24	23, 1	4	4	728	595, 57	46	40, 6	88	41, 38	34	22, 11	28, 2
mālinī	13	13	6	5, 1	209	193, 16	13	6, 7	72	52, 18	11	7, 4	2
pṛthvī					72	67, 4			1	1	5	1, 4	
mandākrāntā					163	136, 8	7	4, 3	3	2	8	6	4
śikhariṇī	3	3			327	309, 18			1		50	31, 15	2, 1
hariṇī					105	99, 6	1	0, 1	1		8	5, 3	3
śārdūlavikrīḍitā					1221	979, 91			5	4	104	73, 27	29, 7
sragdharā					237	181, 24			1	1	21	12, 8	1, 1

Figure 7: *Summary statistics, Overlap (anuṣṭubh, upajāti, samavṛtta), other tools in grey*

(Skr vs. MI and MIT)	Kir		KuS		MSS		RVams		SpV		ST		PT_vv
	1,040		613		9,979		1,637		1,645		324		1,105
ardhasamavṛtta	(MI)	(Skr)	(etc.)										
aparavaktra		1			1	4							
puṣpitāgrā	69	67	2	2	106	99	4	3	78	45	1	1	
viyoginī	62	60	44	44	91	79	97	72	79	71	2	1	1
aupacchandasika	36	33			68	64	2	2	83	68			
viṣamavṛtta													
udgatā	52	38			2	1			91	51			
jāti													
āryā					696	668		1			12	6	67
(MI: āryā "loose")	1				10		1		4		1		
gīti		1			140	144		9		4	1		
upagīti					53	61				1			4
udgīti					35	33							
āryāgīti		4				14		1	1	36			
not identified													
ajñātasamavṛtta		0, 19		0, 1		6, 40		0, 21		0, 128		0, 4	0, 9
ajñātārdhasamavṛtta		1				10		4		51		2	1
no id		1			43	34		5	7	9		19	28

Figure 8: *Summary statistics, Overlap (ardhasamavṛtta, viṣamavṛtta, jāti, errors), other tools in grey*

(Skrutable only)	MBh	Ram	KSS	SS	MM_vv	MMK	BCA	SV	PV_kk	PVA_kk	TS
	77,810	**19,354**	**21,545**	**999**	**226**	**448**	**913**	**3,351**	**1,452**	**2,674**	**3,646**
anuṣṭubh śloka	(Skr)	(etc.)									
pathyā	54811, 591	14576, 235	14528, 50		14	298, 21	638, 13	2985, 152	889, 139	1919, 254	3345, 92
vipulā	17463, 109	3636, 35	6151, 12		1	127, 2	181, 3	149, 5	335, 24	379, 35	201, 2
upajāti											
comm.triṣṭ.	1075	344	51		2		33	9		11	
other triṣṭubh	2162	15		1			3	3		3	
non-triṣṭubh	17, 6	2, 7	2, 4	0, 1	1			0, 1	0, 3	1, 2	0, 2
samavṛtta											
pramāṇikā	38, 1										
indravajrā	174, 87	36, 8	27		8		6	3		2	
upendravajrā	94, 81	80, 15	4		1		2			4, 1	
rathoddhatā	5		10		3					2	
śālinī	38, 62		9		2		0, 1			2	
svāgatā			1							2	
toṭaka											
drutavilambita			1		4			0, 1		2	
pramitākṣarā	0, 3										
vaṃśastha	360, 108	268, 40	26, 3		2			1		3, 1	
praharṣiṇī	8, 1	4	2		7						
mañjubhāṣiṇī					7						
rucirā	28, 13	5, 1									
vasantatilakā	3		122		49		3			2	
mālinī	9		19		21		2	3, 1		1	
pṛthvī			38		4				1		
mandākrāntā			6		14						
śikhariṇī			9		22						
hariṇī					12						
śārdūlavikrīḍitā	1		40		32		1				
sragdharā			1		6		3				

Figure 9: *Summary statistics, Original (anuṣṭubh, upajāti, samavṛtta)*

(Skrutable only)	MBh	Ram	KSS	SS	MM_vv	MMK	BCA	SV	PV_kk	PVA_kk	TS
	77,810	**19,354**	**21,545**	**999**	**226**	**448**	**913**	**3,351**	**1,452**	**2,674**	**3,646**
ardhasamavṛtta											
aparavaktra	27	9			1		2			1	
puṣpitāgrā	33	26	18		5		2				
viyoginī			3		1		3				
aupacchandasika			39		1		9				
viṣamavṛtta											
udgatā											
jāti											
āryā	1	1	168	956	11		1		3	4	
gīti			170	18							
upagīti	46		7	9				5	11	12	
udgīti			1	2			1	1			
āryāgīti										1	
not identified											
ajñātasam.	5, 166	0, 3	0, 16	0, 5	0, 2		0, 1	0, 18	0, 23	0, 8	0, 2
ajñātārdhasam.	51	2			1		2	1		10	
no id	60	1	3	7				16	23	12	2

Figure 10: *Summary statistics, Original (ardhasamavṛtta, viṣamavṛtta, jāti, errors)*

its tightly printing block of "l" and "g" characters, line-by-line syllable and mora counts, and syllable-weight alignment — relies primarily on the visible geometry of the line-separated raw syllable weights, and secondarily on the other scansion information calculated therefrom (morae,

syllable counts, and gaṇas), also presented as line-separated. Since each way has advantages, there is no reason not to eventually offer both as options, at least in a UI.

In the case of anuṣṭubh and upajāti, Skrutable's identification behavior and output labels are uniquely specific. For anuṣṭubh, Skrutable recognizes and names the various vipulās with sensitivity to their usual conditioning (Murthy 2003, Hahn 2014), and it outputs the caveat "not right" (asamīcīna) whenever any conditioning rule is violated, whether it be one of a vipulā or either of the two more fundamental anuṣṭubh rules, "Piṅgala 1" (akṣaras 2 and 3 in odd pāda cannot both be short) and "Piṅgala 2" (akṣaras 2 through 4 in an even pāda cannot be a ra-gaṇa) (Murthy 2003: 103). By contrast, while MI-2015 does recognize the basic vipulā, it uses obscure numerical labels (e.g., "anuṣṭubh3" for a na-vipulā), and it does not explicitly note the role of conditioning. That is, when either of Piṅgala's rules is violated, MIT-2015 fails silently. Meanwhile, MI-2018 seems to be sensitive to none of these details, issuing the label "anuṣṭubh (Śloka)" regardless. Similarly, Skrutable's particular handling of upajātis allows it to distinguish them into subtypes of standard triṣṭubh (indravajrā and upendravajrā), other triṣṭubh (e.g., including vātormī, vāṇī, or also unnamed 11-syllable patterns, e.g. as found in abundance in the Bhagavadgītā), and non-triṣṭubh (the widest meaning of the term "upajāti", as used also for combinations of e.g. 12-syllable i.e. jagatī pādas). Currently, Skrutable takes no further details like āryā syncopation structure (Ollett, 2012) or yatis into consideration.

For the case of imperfect or failed identification, Skrutable's output is again designed to be practical and to avoid both false positives and false negatives. Where some structural pattern is still discernible on the basis of internal support within the verse itself (see §2.2), the suspected type is suggested in the identification label along with a caveat of "unrecognized" ("ajñāta") or "only # pādas correct" ("# eva pādāḥ yuktāḥ") or "not right" ("asamīcīna"), depending on the case. On the other hand, apart from whatever help the basic scansion output may provide for real-time iterative problem-solving, the null identification label ("na kiṃcid adhyavasitam") itself currently provides no specifically helpful information in the following cases: 1) imperfect anuṣṭubhs with zero perfect halves (whether out of one or two), 2) samavṛttas with three or more imperfect pādas, 3) imperfect ardhavṛttas, 4) imperfect viṣamavṛttas, and 5) imperfect jātis.

Here it is useful to consider the alternative strategy of MI-2018, most clearly visible in the case of samavṛttas, to find the nearest identification match using string-alignments of the calculated syllable weight pattern against known patterns (Rajagopalan 2018, pp. 127–128). This allows for extremely clear output in certain cases (see e.g. the red-colored "[-]" in Fig. 11 below, and also the example in Rajagopalan 2018, p. 115).

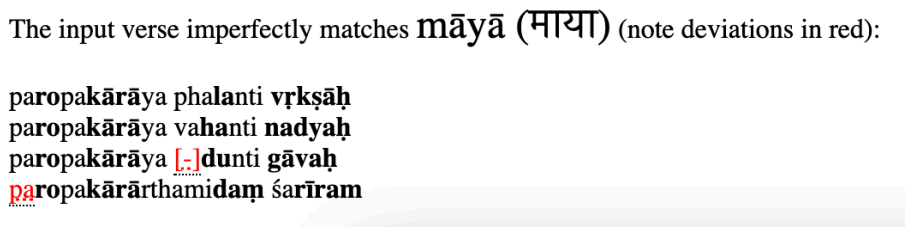


Figure 11: *Screenshot from MI-2018 of samavṛtta alignment output ("dunti" for "duhanti")*

However, this strategy does not appear to work equally well in all cases, and its usefulness is occasionally mitigated by the tendency of MI-2018 (anticipated at Rajagopalan 2018: 132–33) to produce misleading false positives. An example of this is seen in Fig. 11 with the label of "māyā" for an upendravajrā that is missing a syllable in pāda c.[19] Most likely, some combination of these two strategies (reliance on internal structure plus string-alignment), perhaps combined

[19] According to Piṅgala's Chandaḥśāstra (Kedāranātha, 1938), the name "māyā" refers to an upajāti with the specific sequence upendravajrā, upendravajrā, upendravajrā, indravajrā. It is for this reason that the first syllable "pa" of pāda d in Fig. 11 is also mootly colored red as deviating from this "māyā" pattern.

with Skrutable's supplementary scoring strategy — and the latter perhaps also augmented by frequency-based score penalties for very rare meters — may be the best way forward for improving the deterministic approach. Since MI-2018 does also in fact have an unpublished "fulltext" (i.e., whole-file) upload feature in its UI,[20] and since the tool is open source, more systematic comparison of these different output strategies is only a matter of time.

4.3 Speed

I adopt the Tokunaga-Smith digitization of Valmīki's Rāmāyaṇa (19,354 verses) and Ollet's digitization of Weber's Sattasaī/Saptaśatī (999 verses) as speed-run test cases. Comparable speeds can probably also be expected for other tools once they are adapted as necessary,[21] but the main point here is simply to illustrate that text size need no longer pose a significant barrier to such analysis whatsoever.

Rāmāyaṇa

Run on the PythonAnywhere server with the "Re-Split Pādās" option "resplit lite keep mid", the task of analyzing meter for the entire Rāmāyaṇa completes in about 14.5 seconds, i.e., an average of about 1,330 verses per second. With "resplit max keep mid", the time is 23.5 seconds, or about 820 verses/sec, but performance remains nearly identical, with different outcomes only for a dozen cases of triṣṭubh and/or jagatī verse types with data problems. On the other hand, if run on a local machine,[22] "resplit lite keep mid" finishes in about 9.0 seconds, or 2,150 verses/sec, and "resplit max keep mid" again in about 14.5 seconds. If additional explicit pāda break information[23] is used, namely by automatically choosing the relatively confident no-resplit option for such cases and "resplit lite keep mid" for other cases, the time is again reduced to 7.5 seconds, or about 2,580 verses/sec. Again, identification results remain basically constant.

Sattasaī/Saptaśatī

This text, consisting entirely of āryā verses (or because Prakrit, actually gāhā i.e. gāthā), demonstrates the usefulness of Skrutable's "resplit max" option. Because success in identifying āryā is so much better with this option, I do not discuss the "resplit lite" option here. With "resplit max", the successful analysis completes in about 14.5 seconds when run on the PythonAnywhere server (68.9 verses/sec) and in about 10.5 seconds (95.1 verses/sec) when run locally. The slower speed (still less than 1/100th of a second for an individual verse) is due to jāti verses requiring many more resplits before a best match is found.

4.4 Ease of Use

Ease of use is necessarily quite subjective, but some comments are still warranted. With Skrutable, great care has been taken with the UI to reduce effort and possible confusion on the user's part. User focus is maintained on a single display window, which is formatted simply and cleanly with basic Bootstrap CSS and which thus also displays relatively comfortably on mobile devices (see Fig. 1b). Settings allow for users to choose from among numerous transliteration schemes, as well as a range of scansion-detail and other options, all of which are conveniently preserved between sessions. Above all, whole-file processing is available with just a few clicks. In short, user convenience is highly valued, and further feedback is most welcome.

[20] See `http://www.sanskritmetres.appspot.com/fulltext`.

[21] MI-2018 would most accurately be compared by directly utilizing its code. Submission of moderately large texts to its unpublished "fulltext" website endpoint so far suggests speeds slower by one order of magnitude, whereas processing seems to hang in the case of very large texts of 10,000 verses or more (likely a file-size constraint of the server). Inspection of output is also complicated by this mode's emphasis on complex HTML formatting, which makes output rendering prohibitively slow.

[22] Here, on a MacBook Pro 2020 with 2 GHz Quad-Core Intel Core i5. The Jupyter Notebook used for local runs is also available for inspection at `https://www.github.com/tylergneill/skrutable/tree/master/jupyter_notebooks`.

[23] E.g., such as was formerly available on GRETIL at least as late as December 2018 in the form of semicolons between ab and cd pādas.

5 Conclusion

With digitized texts and capable computing resources becoming ever more available, it is high time that students and scholars of Sanskrit be able to use a tool to analyze the metrical properties of a given text without themselves needing to know how to code. Skrutable empowers users to do just that, for any number of verses at a time, accurately, with detailed output, at speed, and (in good cases) with only minimal data prep. As a bonus, those who do code and who want to include such scansion or meter-identification functionality in their own computational projects can also take advantage of Skrutable's being written modularly and in a popular language with robust community support. What's more, Skrutable provides a number of additional practical features, which e.g. may help users better understand certain kinds of imperfect metrical Sanskrit text.

Going forward, Skrutable has room to grow. As an open-source project, others can take it and develop versions as they wish. As for the master branch of its development, as more thorough comparison with similar tools more clearly reveals relative strengths and weaknesses, advantageous features, whether pertaining to core algorithm or to final display, can be either incorporated centrally or included as additional options. For example, MI-2018 makes numerous practical advances yet to be emulated in Skrutable, including the illustrative highlighting of erroneous syllables, and it seems clear to me that even the online version of MIT-2015 (not to mention the further developments thereof in TEITagger) is much more accurate and detailed with regard to certain kinds of good data. That said, even as-is, Skrutable can already make a useful contribution. Alongside its pedagogical value, it can help specialists to not only augment their own research but also contribute annotated material toward a collaborative meter dataset, begun here with 150,000 verses. Such aggregated data can serve both as an interesting reference tool in its own right and as a model-training resource for very exciting technological applications to come.

Acknowledgements

The author's thanks are due to Peter Scharf and The Sanskrit Library, for a formative period of part-time employment cataloging Sanskrit manuscripts at Houghton Library in 2015 and for sharing software and data; to Shreevatsa Rajagopalan, who in a personal communication permitted reference to his unpublished table of summary statistics and to the "fulltext" endpoint of his MI-2018 website; and to Oliver Hellwig and Sebastian Nehrdich, for allowing me to feature their Splitter.

References

Ashwini S. Deo. 2007. The Metrical Organization of Classical Sanskrit Verse. *Journal of Linguistics*, 43(1):63–114.

Michael Hahn. 2014. A brief introduction into the Indian metrical system (for the use of students). Handout. Accessed at `academia.edu/6353023` on Mar. 31, 2021.

Oliver Hellwig and Sebastian Nehrdich. 2018. Sanskrit Word Segmentation Using Character-level Recurrent and Convolutional Neural Networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763, Brussels, Belgium. Association for Computational Linguistics.

Daniel H. H. Ingalls. 1985. The "Mahābhārata": Stylistic Study, Computer Analysis, and Concordance. *Journal of South Asian Literature*, 20(1):17–46.

Pt. Kedāranātha. 1938. *Śrī-Piṅgalanāga-viracitaṃ Chandaḥ-śāstram: Śrī-Hālāyudha-bhaṭṭa-viracitayā Mṛtasaṃjīvanyākhyayā vṛttyā sametam. Samīkṣācakravarti-rāja-paṇḍita-śrī-Madhusūdanavidyāvācaspati-racita-Chandonirukti-sanāthī-kṛtaṃ ca = The Chhandas-śāstra of Piñgalanāga.* Nirnaya Sāgar Press, Bombay, 3rd edition.

Keshav Melnad, Peter Scharf, and Pawan Goyal. 2015. Meter Identification of Sanskrit Verse. In *Sanskrit Syntax: Selected Papers Presented at the Seminar on Sanskrit Syntax and Discourse Structures, 13-15 June 2013, Université Paris Diderot, with an Updated and Revised Bibliography by Hans Henrich Hock*, pages 325–346. Sanskrit Library, Providence.

G. S. Srinivasa Murthy. 2003. Characterizing Classical Anuṣṭup: A Study in Sanskrit Prosody. *Annals of the Bhandarkar Oriental Research Institute*, 84:101–115.

Andrew Ollett. 2012. Moraic Feet in Prakrit Metrics: A Constraint-Based Approach. *Transactions of the Philological Society*, 110(2):241–282.

S. Rajagopalan. 2018. A user-friendly tool for metrical analysis of Sanskrit verse. In *Computational Sanskrit & Digital Humanities, Selected papers presented at the 17th World Sanskrit Conference, July 9-13, 2018*, pages 113–142, University of British Columbia, Vancouver, Canada. Department of Asian Studies, University of British Columbia.

Rama N. and Meenakshi Lakshmanan. 2010. A Computational Algorithm for Metrical Classification of Verse. *International Journal of Computer Science*, 7(2):46–53, March.

Peter Scharf. 2018. TEITagger: Raising the standard for digital texts to facilitate interchange with linguistic software. In *Computational Sanskrit & Digital Humanities, Selected papers presented at the 17th World Sanskrit Conference, July 9-13, 2018*, pages 229–257, University of British Columbia, Vancouver, Canada. Department of Asian Studies, University of British Columbia.

Sven Sellmer. 2013. Computer-aided detection of unusual passages in the Mahābhārata. In *CEENIS Current Research Series*, volume 1, pages 44–54. Dom Wydawniczy Elipsa, Warsaw.

John D. Smith. 1999. Winged Words Revisited: Diction and Meaning in Indian Epic. *Bulletin of the School of Oriental and African Studies, University of London*, 62(2):267–305.