Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 7069–7076
Marseille, 11–16 May 2020

SHR++: An Interface for Morpho-syntactic annotation of Sanskrit Corpora

Amrith Krishna[★], Shiv Vidhyut[✦], Dilpreet Chawla[●], Sruti Sambhavi[♣], Pawan Goyal[✿]
[★]IT University of Copenhagen, [✦]IIITDM Kancheepuram, [●]IIIT Kalyani
[♣]NIT Rourkela, [✿]IIT Kharagpur
amrith@iitkgp.ac.in, pawang@cse.iitkgp.ac.in

Abstract

We propose a web-based annotation framework, SHR++, for morpho-syntactic annotation of corpora in Sanskrit. SHR++ is designed to generate annotations for the word-segmentation, morphological parsing and dependency analysis tasks in Sanskrit. It incorporates analyses and predictions from various tools designed for processing texts in Sanskrit, and utilises them to ease the cognitive load of the human annotators. Specifically, SHR++ uses Sanskrit Heritage Reader (Goyal and Huet, 2016), a lexicon driven shallow parser for enumerating all the phonetically and lexically valid word splits along with their morphological analyses for a given string. This would help the annotators in choosing the solutions, rather than performing the segmentations by themselves. Further, predictions from a word segmentation tool (Krishna et al., 2018) are added as suggestions that can aid the human annotators in their decision making. Our evaluation shows that enabling this segmentation suggestion component reduces the annotation time by 20.15 %. SHR++ can be accessed online at `http://vidhyut97.pythonanywhere.com/` and the codebase, for the independent deployment of the system elsewhere, is hosted at `https://github.com/iamdsc/smart-sanskrit-annotator`.

Keywords: low-resource languages, Sanskrit, annotation, dependency trees, morphological analysis, word segmentation, visualization, GUI, crowd-sourcing, treebanking

1. Introduction

Sanskrit is a classical language (Coulson, 1992; Sapir, 2004), and used to be the 'lingua franca' for the scientific, literary and philosophical discourse in the Indian subcontinent for more than 3 millennia. The language has over 30 million extant manuscripts potent for digitisation, one hundred times those in Greek and Latin combined (Goyal et al., 2012). However, a major source of concern for the processing of texts in a low-resource language like Sanskrit is the lack of availability of labelled data. Acquiring task-specific annotations can be linguistically involved and time consuming, making it a challenge in itself. In this work, we present 'SHR++', a web-based annotation framework that would assist a professional annotator[1] to perform annotations for three different tasks, ranging from word-segmentation, morphological parsing and dependency parsing.

Word-segmentation is a challenging task in Sanskrit, as the writings in Sanskrit follow a phonemic orthography. Here the words in the sentence often undergo phonetic transformations at the juncture of the word boundaries, which not only obscure the word boundaries but also modify the graphemes present in the string. This euphonic assimilation of phones is formalised as *Sandhi*. *Sandhi*, while mostly deterministic in generation, is strongly ambiguous in analysis. This makes segmentation more difficult than languages such as Chinese, where the words are combined without any euphonic assimilation at the boundary (Goyal and Huet, 2016). In this work, we extend the functionalities of a lexicon-driven shallow parser and annotation framework for Sanskrit, Sanskrit Heritage Reader (Goyal and Huet, 2013), henceforth to be referred to as SHR. It uses finite-state methods in the form of a lexical juncture system to obtain the possible segments for a given sentence (Goyal and Huet, 2016; Huet and Goyal, 2013). Here, SHR enumerates all the valid segmentation candidates for a given sequence along with their possible morphological analyses. Figure 1 shows one such analysis for a given sequence (as displayed in our proposed system SHR++). Here, the human annotator's job is to choose the correct candidate words and the morphological analyses for those selected candidates based on the output from SHR. Given the extensive presence of syncretism and homonymy in Sanskrit (Krishna, 2019; Sims, 2015), this is performed as a two step process. The human annotator first selects the segmentation candidates and then chooses the appropriate morphological tag for the selected candidates. SHR++ not only incorporates all the annotation functionalities of SHR, but it is an improvement from SHR in the following three significant ways:

1. In SHR++ we extend the system to include annotation capabilities for dependency relations, in addition to the annotation of word segmentation candidates and their morphological tags. Given that the morphology and syntax are tightly interlinked in a language like Sanskrit (Kiparsky, 1994), our framework still uses only a two step approach for the annotations. In the first step, we replicate the working of SHR, while in the second step we expect the user to annotate the dependency relations, along with the corresponding morphological tags using our user-friendly and intuitive interface.

2. Our framework facilitates the incorporation of predictions from automated word segmentation tools for Sanskrit (Krishna et al., 2018) and the use of these predictions as suggestions to the human annotators. Through a human judgement experiment, we find that using these predictions as suggestions results in an average of 20.15 % reduction in the annotation time for the human annotators.

[1] A person proficient with Sanskrit. Though Sanskrit currently has less than 15,000 documented native speakers, it is widely offered as a linguistics course in more than 1000 institutions in India alone. We assume our annotator to have acquired a formal education in Sanskrit Linguistics.

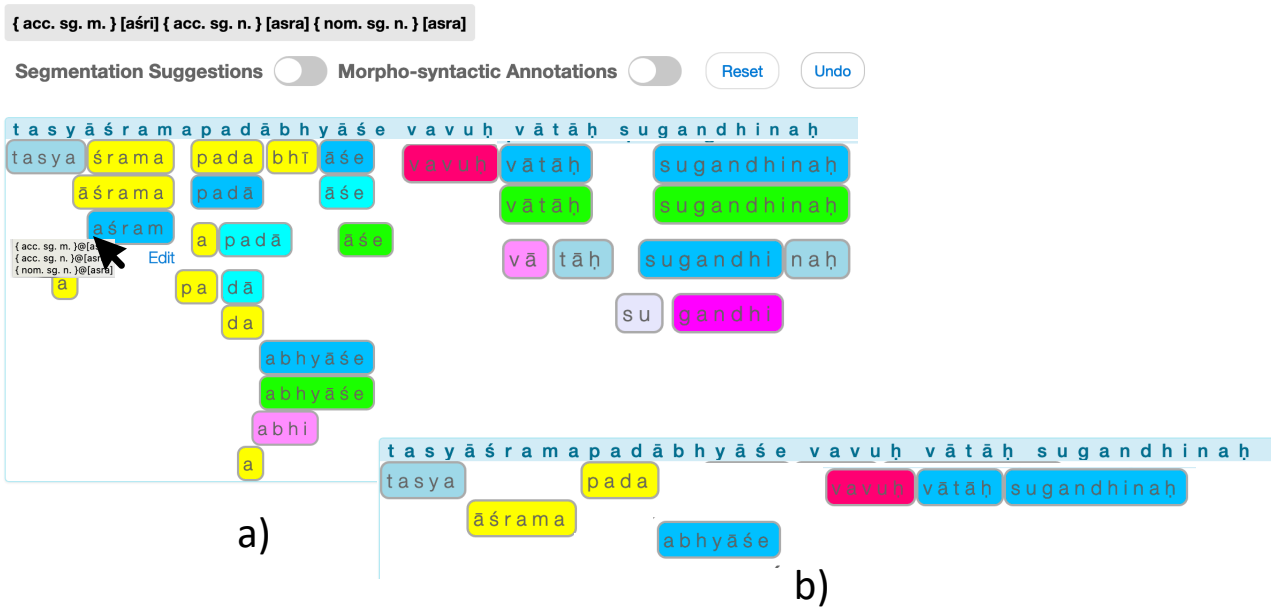


Figure 1: a) Segmentation analysis and the b) correct solution for the sequence 'tasyāśramapadābhyaśe vavuḥ vātāḥ sugadhinaḥ' as displayed in the SHR++ interface. In the figure, the mouse pointer is hovered over the candidate word aśram. Due to this, the morphological analysis for this word is shown both as a tooltip near the word and on the top portion of the interface. The analysis for the given sentence can be accessed at `http://bit.ly/shrpp`.

3. SHR allows for accessing the analyses and the annotations only via its web interface. It does not provide a means to export the necessary data into any of the standard formats. SHR++ facilitates storage and export of data into standard data formats, including CSV, JSON, SQL or even in CoNLL-X format, the widely used format for storing dependency annotations. Additionally, we also provide a web interface to access and manage the data that was annotated using the system.

2. Background

The usage of Sanskrit was strongly rooted in its oral tradition (Staal, 2008) prevalent during the pre-classical and classical time period in the Indian subcontinent. This has shaped the characteristics of the language greatly, with many of those characteristics reflected in the writing as well. The euphonic assimilation of phones at the juncture of the words, similar to what one observes in connected speech, is one such feature. The presence of an advanced discipline of phonetics in Sanskrit formalises this euphonic assimilation of phones as *Sandhi* (Goyal and Huet, 2016). *Sandhi*, while mostly deterministic during generation, leads to ambiguities during analysis. While such transductions are commonly observed between morphemes of a word in various languages, here the transformations happen between words in a sentence. Figure 1 shows the segmentation analysis and the final segmented solution for the sentence '*tasyāśramapadābhyaśe vavuḥ vātāḥ sugadhinaḥ*'. Here, consider the substring from the given sentence, '*tasyāśrama*'. It can be split in four different ways, i.e. '*tasya* + *śrama*', '*tasya* + *āśrama*', '*tasya* + *a* + *aśram*' or as '*tasya* + *aśram*' by virtue of *Sandhi*. The last 3 cases show phonetic transformations, resulting in change of 'ā' to 'a + ā', 'a + a + a' and 'a + a' respectively. Figure 1a, shows the various candidate segments for the given sentence and the sentence can have 2446 different segmentation solutions.[2] The correct solution, along with the splits for the compound components (shown in yellow), are shown in Figure 1b. Given the extensive use of multi component compounds and the use of compounds in place of the syntactic co- and sub-ordination in Sanskrit (Lowe, 2015), word segmentation in Sanskrit typically involves splitting of compounds into its components as well (Goyal and Huet, 2016; Hellwig and Nehrdich, 2018).

The word-splits in Figure 1a are obtained based on the analysis from SHR. SHR uses finite state methods in the form of a lexical juncture system to obtain the possible segments for a given sentence. Following definitions from Goyal and Huet (2016)[3], a *lexical juncture system* can be formalised as follows. It is composed of a finite alphabet Σ, a finite set of words $L \subseteq \Sigma^*$ and a finite set R of rewrite rules of the form $u|v \rightarrow f/x\_\_$ (Kaplan and Kay, 1994), with $x, v, f \in \Sigma^*$ and $u \in \Sigma^+$. Here, Σ forms the set of phonemes, R is the set of Sandhi rewrite rules, and L is the vocabulary which is a set of lexical entries. Each entry $z \in L$ is defined as a tuple (l, m, w). Here l denotes the lemma of the word, m denotes the morphological class of the word, w denotes the inflected surface-form generated from l and m. The sandhi analysis S_i for a string s can be seen as a sequence $\langle z_1, \sigma_1, k_1\rangle; ...\langle z_p, \sigma_p, k_p\rangle$. Here, $\langle z_j, \sigma_j, k_j\rangle$ is a segment with $z_j \in L$, k_j denotes the position at which the word w_j begins in the sentence s and $\sigma_j = x_j u_j | v_j \rightarrow f_j \in R$ for $(1 \leq j \leq p)$, $v_p = \epsilon$ and $v_j = \epsilon$ for $j < p$ only if $\sigma_j = o$, subject to the matching conditions: $z_j = v_{j-1} y_j x_j u_j$ for some $y_j \in \Sigma^*$ for all $(1 \leq j \leq p)$, where by convention $v_0 = \epsilon$. Finally $s = s_1 ... s_p$ with $s_j = y_j x_j f_j$ for $(1 \leq j \leq p)$, ϵ denotes the empty word. Further, words

[2] Estimated based on the number of syncretisms and homonyms involved in the analysis.

[3] We recommend the readers to refer to the work for a detailed review of SHR.

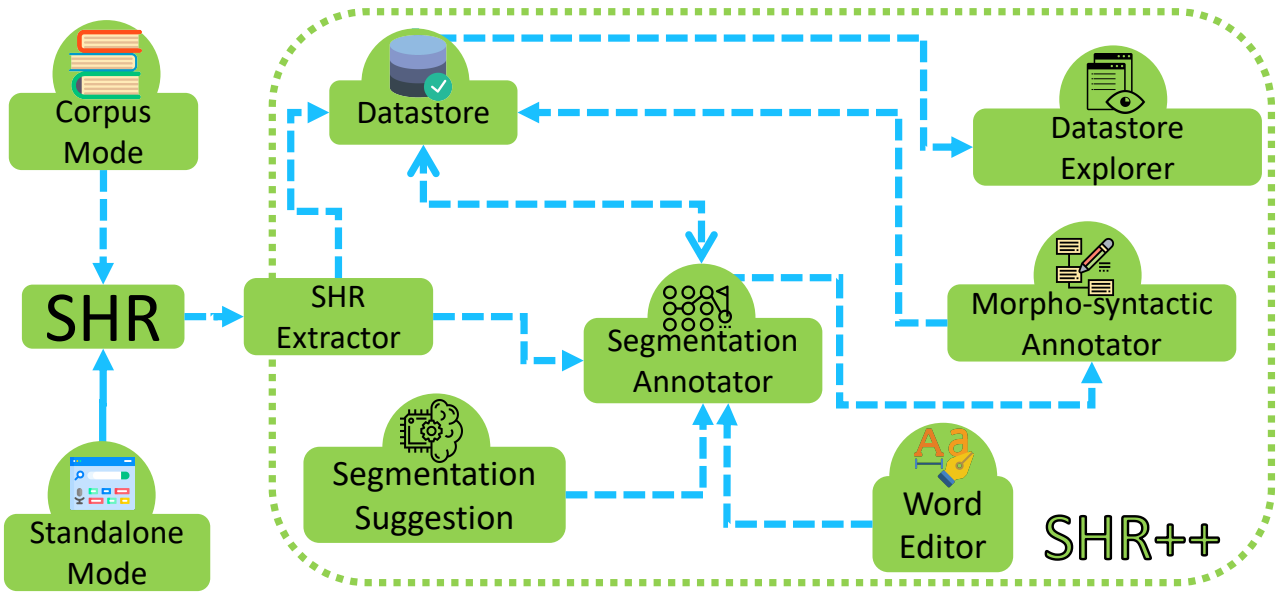


Figure 2: Overview of the modules in SHR++ and their interactions within the system.

that are proposed as alternatives to each other, and hence cannot co-exist in a single solution, are called as 'conflicting' word-pairs. The words 'śrama', 'āśrama', 'aśram' are conflicting to each other.

Sanskrit is a morphologically rich language and consists of a rich tag-set of 1,635 possible tags for its inflectional morphology (Krishna, 2019; Hellwig, 2016). Further, cases of syncretism and homonymy are prevalent for the fusional language (Krishna et al., 2018; Hellwig, 2015b). SHR, similar to its segmentation analysis, also performs the possible morphological analyses for each of the candidate words. For a compact representation of the candidate words, both SHR and SHR++ display only the unique inflected forms in the segmentation analysis and then incorporate all the valid morphological analyses for the surface-form within the surface form itself. This information for each word is shown to the user on hovering the mouse pointer over the corresponding word. The morphological analysis for the word 'aśram' is shown in Figure 1a. Here, the surface-form 'aśram' essentially consists 3 valid analyses, of which 2 are cases of syncretism for the stem 'asra' and the third one is a case of homonymy where stem for the inflection is 'aśri'. The identification of the correct morphological tag is expected only at the second phase of annotation in both SHR and SHR++.

The second phase of annotation in SHR++, expects the human annotator to annotate the morphological information and the syntactic (dependency) relations jointly. This decision of ours is motivated in principle from the traditional grammatical treatise on Sanskrit, *Ashtādhyāyī*. *Ashtādhyāyī*, written by *Pāṇini* about 2500 years ago, treats the morphology and syntax as a single subsystem (Kiparsky, 1994). It uses the *Kāraka* framework for the analysis of a sentence, which stands close to the dependency analysis of sentences followed in western linguistics (Begum et al., 2008; Bharati et al., 2019). The *Kāraka* relations, formed between various word pairs in a sentence, are syntactico-semantico relations between the verbals and other related constituents in a sentence, with the analysis resulting in a dependency tree (Bharati and Sangal, 1993). *Kāraka* relations are pivotal in the assignment of case and other morphological elements for the words in a sentence. "The key principle is that every *Kāraka* relation must be expressed by a morphological relation, and none can be expressed by more than one" (Kiparsky, 1994).

3. System

Our system, SHR++, has a client-server architecture, where the server component is completely built using Django[4], a python framework. The client-side uses a web-based front-end (HTML+CSS+JS). Figure 2 shows the overview of SHR++ and the various modules it contains. Our system is designed to work in two different modes, the corpus mode and the standalone mode. In the standalone mode, we provide a web-interface that enables the user to input a string (a word, phrase, fragment or a sentence) in Sanskrit. The user input is then passed onto SHR in real-time for enumeration of the possible word-splits and their morphological analyses. In corpus mode, we expect that the analyses for all the sentences in a corpus are already populated into a datastore, via a command-line script from the back-end itself. This facilitates bulk upload of data. Apart from this, both modes work exactly the same. Sentence level analysis and the resulting candidates are then displayed to the human annotator, as shown in Figure 1. Here, we expect the annotator to identify the correct segmented word forms and then proceed for the morpho-syntactic analysis of the selected word forms, as shown in Figure 3. Now we detail, each of the component involved in SHR++.

Segmentation Annotator: As shown in Figure 1a, the segmentation candidates for a given string are presented to the annotator. The challenge here is that the number of

[4] https://www.djangoproject.com/

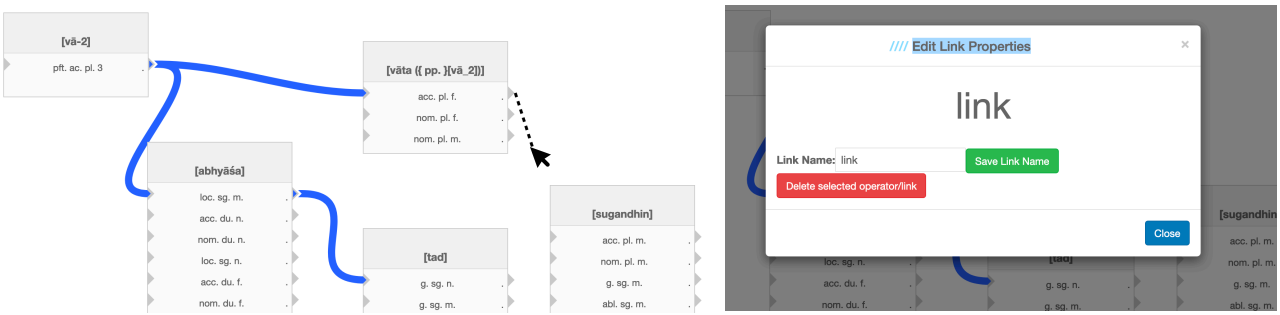


Figure 3: Interface for the morpho-syntactic annotator and the overlay 'lightbox' interface for labelling the relations

candidates can exponentially increase with the length of the string and aligning these candidates to fit in a screen can be a challenge in itself (Goyal and Huet, 2016). Here, we faithfully follow the design scheme adopted by SHR to represent the candidate word-splits. SHR uses the term 'shared forest' (Goyal and Huet, 2016) to refer to such an arrangement. However, this is similar to that of a 'lattice' used in the lattice based morpho-syntactic parsing approaches (Kudo et al., 2004; More et al., 2018; Seeker and Çetinoğlu, 2015) proposed for languages such as Korean, Hebrew, Turkish etc. The segmentation annotator forms first phase of annotation, where only the unique inflected word-forms are shown to annotators. Among the candidate words, the longer ones are placed in the topmost row, provided none of them conflict with each other. This policy is followed for every subsequent row and leads to a compact display. When a user clicks on the word, its corresponding conflicting candidates are eliminated automatically. Clicking back on the word would undo this selection, and previously eliminated conflicting words are displayed back for the un-selected word. All the possible morphological analyses for a given candidate word are shown by hovering the mouse over it. As stated previously, we only focus on annotating the surface-form level segmentations with this module. Further, we colour-code the words based on their morphological classes. For instance, we use blue for substantives, red for finite verbforms, purple for adverbs, pale blue for pronouns and yellow for compound components (Goyal and Huet, 2016). Since the final component of compound in Sanskrit carries the inflectional markers, it is represented with blue color.'aśramapadābhyāśe' is one such compound word with the correct splits shown in Figure 1b.

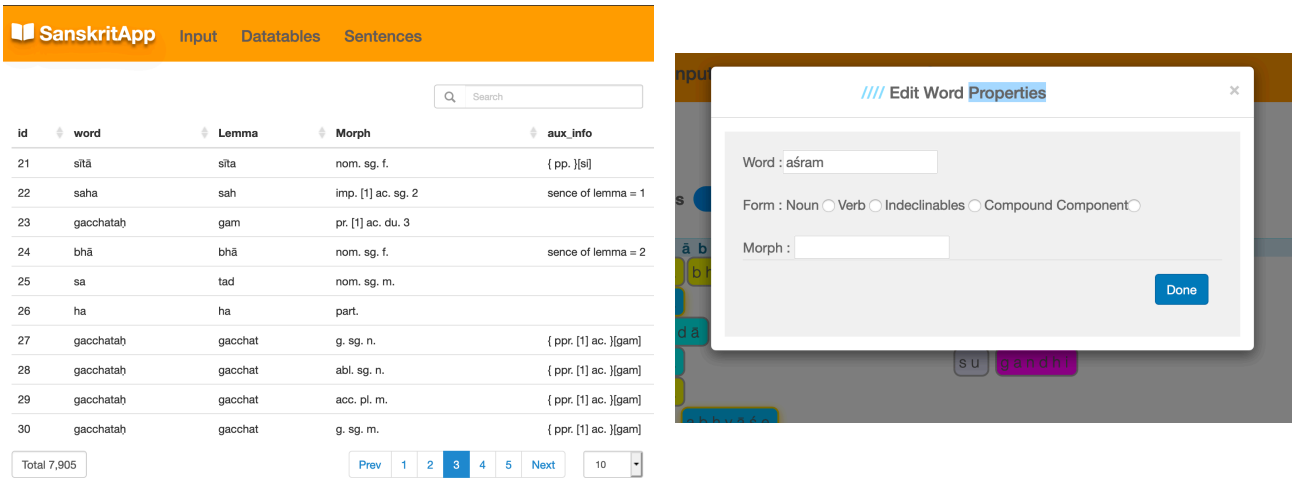
Segmentation Suggestion: In addition to showing the segmentation candidates from SHR, we also facilitate the incorporation of predictions from automated word-segmentation tools designed for Sanskrit (Krishna et al., 2018; Hellwig and Nehrdich, 2018) as suggestions to the human annotators. Based on the predictions from these systems[5], the relevant candidate words are highlighted so as to assist the annotator by inviting her attention to those words. However, the final decision still remains with the annotator, who may or may not choose the suggestion. The annotators may enable or disable this feature by clicking on the 'Segmentation Suggestions' button as shown in Figure 1a. Currently, this feature is intended to be used only in the corpus mode, where the predictions from the automated segmenters are obtained offline and stored in our datastore. This is primarily due to the time constraints required for obtaining the predictions in real-time using the inference procedure of the word-segmentation tools.

Word Editor: It is possible that SHR may not produce an analysis for certain substring-spans of the input string. It can also be possible that the analyses provided by SHR for some of the candidate words might be erroneous and require correction (Krishna et al., 2017). Taking into consideration such cases, we add a provision for the annotators to modify the candidate word analyses. On hovering the mouse over a candidate word, an edit option appears below it. It can be used by the human annotator to edit the details of the word. As shown in Figure 4b, the annotator can edit the surface form, the lemma as well as the morphological tags associated with the words. In case of missing analysis for a span of the input, a dummy candidate word is provided beneath the span, which the annotator can edit to add the appropriate words. All the modifications made by the human annotators get stored in our datastore.

Morpho-syntactic Annotator: This forms the second phase of analysis, where we expect a human annotator to perform morphosyntactic annotations, i.e. morphological information and dependency relations, for the candidate words selected using the segmentation annotator. In SHR, the annotator is guided to a new page once the annotator completes the identification of all the segmentation candidates for a given string. In SHR++, both the phases can be accessed from the same screen. Even with a partial selection of candidates in the segmentation annotator, the morphosyntactic annotator can be accessed by enabling the "Morpho-syntactic Annotations" button. This button is placed above the segmentation annotator component and is shown in Figure 1a. This gives the annotator the freedom to switch between both the phases of annotations as per her will. Figure 3 shows the interface for the annotation of morphological and syntactic information. The interface uses the jQuery flowchart module[6], a jQuery UI plugin originally designed for drawing flowcharts. Each surface-form candidate, selected during segmentation annotation, is represented as a separate node in the flowchart interface. Every

[5] Currently, we incorporate predictions only from Krishna et al. (2018). But this can be trivially extended to other prediction systems as well.

[6] http://sebastien.drouyer.com/jquery.flowchart-demo/



a) Datastore b) Word editor

Figure 4: a) An instance of the datastore explorer component. b) An instance of the word-editor component.

node, i.e. a surface-form, has at least one flow, i.e. one inflow and one outflow, and the number of flows equals the number of morphological analyses each surface-form has. The human annotator has to construct the edges between words that are related by a syntactic relation. In this process, the flow corresponding to the relevant morphological tag needs to be selected. This automatically disables all other flows for the node, as they were suggested as alternatives to each other, and hence cannot co-exist in a sentence. Once an edge is created, the human annotator may click on it to add the dependency relation as the edge label. A 'lightbox'[7] interface, as shown in Figure 3, is used to capture the edge label information.

SHR Extractor and Datastore: SHR provides only an HTML output of its analyses for the input strings. We build a scraper, that would scrape this information from SHR's HTML output and convert it into a structured tabular format. It needs to be noted that, SHR makes several design decisions to represent syncretisms, homonyms, compound splits and derivational nouns (from verbs), which we take care of during our scraping procedure and this makes scraping a non-trivial exercise. In the independent mode, the user is supposed to pass one sentence at a time via the interface, while in the corpus mode the user can use a single command to pass multiple sentences in bulk. In either of the cases the sentences are parsed by SHR. In the corpus mode, the SHR analysis is saved onto the datastore directly and we use SQLite[8] databases for the purpose. In the standalone mode, the SHR analysis is directly passed to the segmentation annotator to avoid any latency, and then later saved onto the datastore. For in-memory handling of the data, we use pandas dataframes (McKinney, 2010). SHR++, by virtue of using pandas, enables for import and export of the data into various standard structured data formats, including JSON, CSV, XML etc. Further, we also facilitate exporting the data into a revised version of CoNLL-X format. CoNLL-X and its variants are a widely adopted format for storing dependency annotations. Additionally, we add provisions for handling file formats such as GraphML in which other popular Sanskrit datasets are released (Krishna et al., 2017).

Datastore Explorer We also provide a web based interface to explore the SQLite database which contains the SHR analyses and annotations for all the sentences that have been populated in the database. This is an additional feature from SHR, and allows for corpus level exploration of the data. Figure 4a shows an instance of the datastore explorer.

4. Evaluation

We perform a human annotator evaluation to evaluate the efficacy of our proposed system. Table 1 shows the results for the evaluation. Here, we compare whether there exists any significant difference in time taken for annotations, when using SHR and SHR++. We also evaluate whether the use of segmentation suggestion module in SHR++ leads to any significant reduction in annotation time. Finally, we also report the time taken for morphosyntactic annotation in SHR++. The annotations were performed by 3 human experts, each with at least an undergraduate degree in Sanskrit Linguistics.

We randomly selected 60 textlines from the digital corpus of Sanskrit (Hellwig, 2010 2016). As our objective of the evaluation was to compare the time taken for performing the annotations, we only considered those sentences in which the human annotators made no mistakes during the annotation. This resulted in a final set of 30 sentences. By the end of the experiment, each annotator ends up annotating all the 30 sentences. However, an annotator annotates a particular sentence for only one of the 3 annotations tasks, i.e. segmentation annotation without suggestion, segmentation annotation with suggestions and morphosyntactic annotation. By this, we make sure that the annotators do not perform two annotation tasks on the same sentence, as familiarity with the sentences may affect the annotation time. For segmentation annotation using SHR, 3 new expert annotators were asked to perform the annotations on the same set of sentences.

The 30 sentences in the final set were further divided and grouped into 10 different sets, each containing 3 sentences.

[7] http://fancybox.net/

[8] https://en.wikipedia.org/wiki/SQLite

Sl. No.	Sentence Length	Total Solutions	Number of correct Splits	Segmentation annotation	Segmentation annotation with suggestion	Morphosyntactic annotation
1	80 - 95	3839	6.67	31.67	26.34	54
2	96 - 110	6418.33	8.67	30.67	25.9	58
3	111 - 125	9017.67	8	34.67	23.01	46.67
4	126 - 140	16814.33	12.33	39.33	32	59.67
5	141 - 150	417295.66	14.33	43	31.67	69.33
6	151 - 165	19781779	16.67	55.67	42.34	86
7	166 - 170	47246874.33	15	49.33	39.67	84.33
8	171 - 185	100276799	18.67	71.67	58.34	102
9	186 - 200	72674493.66	22.33	88.33	68.66	113.67
10	201 - 215	103810590	22	95	82.67	129
Avg.				53.934	43.06	80.267

Table 1: Evaluation in terms of time taken for morphosyntactic annotation and segmentation annotation (with and without suggestions) of Sanskrit sentences. Each row represents the values averaged over sets of 3 sentences each.In the last three columns, the time taken is reported in seconds.

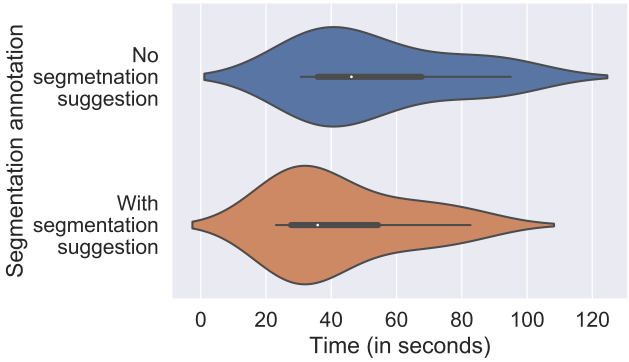


Figure 5: Violin plots showing the distributions of time taken for segmentation annotation phase with and without the segmentation suggestion component. The white dot in the middle denotes the median value and the thick black bar in the centre represents the interquartile range.

The sentences are grouped based on the number of characters present in them.[9] Table 1 shows the number of possible segmentation suggestions that are available for the sentences as well as the number of correct splits in the final solution. It also shows the average time taken for the annotation task per set. From the violin plot shown in Figure 5, it can be observed that enabling the segmentation suggestion component consistently results in lesser annotation time as compared to the setting where the component is disabled. The set-wise macro-averaged time taken to complete the annotations with and without the segmentation suggestion component is 43.06 and 53.93 seconds, respectively. It can be observed that the use of segmentation suggestion component leads to a 20.15 % reduction in the time taken to perform the annotations. For the experiment, the suggestions are obtained using the word segmentation tool proposed by Krishna et al. (2018). The tool reported a macro-averaged F-score of 96.92 % on their test data and it reports 94.21 % for the 30 sentences we use in the experiment. Similarly, Table 1 also shows the time taken for the morphosytnactic annotation and reports an average of 80.27 seconds for this phase. We could not observe any significant differences in the annotation time taken for segmentation annotation in both the SHR and SHR++. The average time taken in SHR is 56.8 seconds as compared to the 53.93 seconds in SHR++.

5. Discussion and Future Work

SHR++ is a Django based annotation framework. It is designed to be independent in its setup and functioning from that of SHR, even though we rely on SHR for the analysis of the input strings. Currently, SHR++ obtains the analysis for the input strings by relying on the SHR web-service hosted online.[10] We design our own scraping tool to obtain the data from SHR. SHR++ is designed primarily to be an annotation interface which can accept data from various sources and structured data formats. This gives us a leverage that our annotation framework can be easily used along with other numerous tools (Kulkarni and Ramakrishnamacharyulu, 2013; Susarla and Challa, 2019; Hellwig, 2015a) and datasets (Krishna et al., 2017) available for Sanskrit. In fact, SHR++ can be used as an annotation framework for any language, such as Korean, Turkish, Hebrew etc., that is amenable to a lattice-based parsing approach (Hatori et al., 2012; More et al., 2019; Kudo, 2006; Smith et al., 2005). At the same time, using a scraper to obtain the analyses from SHR leads to latency issues and considerable performance overhead. This is particularly true in the case of standalone mode, where the analysis from SHR is required real-time, based on the query from the user. But, we leave this for future.

Yet another reason to decouple SHR from SHR++ is due to the differences in the frameworks and languages used in development of both the systems. SHR is implemented completely in OCaml for its analysis, and uses a CGI based interface to interact with its web interface. At the same time, SHR++ is written purely using the Django framework. Our decision to use a python based framework stems from our

[9] number of characters are decided as per the SLP1 encoding scheme (Scharf and Hyman, 2011).

[10] https://sanskrit.inria.fr/DICO/reader.fr.html

plans to extend the functionalities of SHR++ in the future. Currently the segmentation suggestion component utilises the segmenter only from Krishna et al. (2018). We plan to extend this to other word segmentation tools for Sanskrit (Hellwig and Nehrdich, 2018; Aralikatte et al., 2018; Reddy et al., 2018). Similar in spirit to CROWDTREE (Tratz and Phan, 2018), the annotation framework can be used for developing a human in the loop machine learning system, where the annotations and feedback from the annotators can be used as inputs to an active learning framework or more specifically, to a co-active learning framework (Shivaswamy and Joachims, 2012). This enables us to leverage the use of logical constraints from the traditional grammatical framework of Sanskrit, the wisdom of the crowd and the statistical inference power of machine learning.

6. Conclusion

In this work, we propose an online annotation tool for annotating sentences in Sanskrit for three tasks, namely, word-segmentation, morphological parsing and dependency parsing. While there have been tremendous advances in digitising texts in Sanskrit in the past 2 decades (Hellwig, 2010 2016; Scharf, 2009), there still exist limitations in the availability of task-specific labelled data (Goyal et al., 2012). We believe SHR++ will be yet another step in bridging the gap in this area. Summarily, our tool extends the SHR so as to facilitate annotation of syntactic relations and also to enable storage and sharing of data using standard structured data formats. Our tool also enables to ease the cognitive load of the annotator, by making use of predictions from automated word segmentation tools as suggestions. Our evaluation experiment shows that this results in a reduction in annotation time by about 20.15 %.

Tool and codebase: Our annotation tool can be accessed online at `http://vidhyut97.pythonanywhere.com/` and our codebase along with installation instructions can be accessed at `https://github.com/iamdsc/smart-sanskrit-annotator`.

Acknowledgements

We are grateful to Gérard Huet for providing the Sanskrit Heritage Engine. We extend our gratitude to Amba Kulkarni, Oliver Hellwig, Peter Scharf and Najmeh Abiri, along with Gérard for helpful comments and discussions regarding the work.

7. Bibliographical References

Aralikatte, R., Gantayat, N., Panwar, N., Sankaran, A., and Mani, S. (2018). Sanskrit sandhi splitting using seq2(seq)2. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4909–4914, Brussels, Belgium, October-November. Association for Computational Linguistics.

Begum, R., Husain, S., Dhwaj, A., Sharma, D. M., Bai, L., and Sangal, R. (2008). Dependency annotation scheme for Indian languages. In Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II.

Bharati, A. and Sangal, R. (1993). Parsing free word order languages in the paninian framework. In 31st Annual Meeting of the Association for Computational Linguistics, pages 105–111, Columbus, Ohio, USA, June. Association for Computational Linguistics.

Bharati, A., Kulkarni, A., and Sharma, D. M. (2019). Pāṇinian syntactico-semantic relation labels. In Proceedings of the Fifth International Conference on Dependency Linguistics, DepLing 2019 (Accepted).

Coulson, M. (1992). Sanskrit: An introduction to the classical language.

Goyal, P. and Huet, G. (2013). Completeness analysis of a sanskrit reader. In Proceedings, 5th International Symposium on Sanskrit Computational Linguistics. DK Printworld (P) Ltd, pages 130–171.

Goyal, P. and Huet, G. (2016). Design and analysis of a lean interface for sanskrit corpus annotation. *Journal of Language Modelling*, 4(2):145–182.

Goyal, P., Huet, G., Kulkarni, A., Scharf, P., and Bunker, R. (2012). A distributed platform for Sanskrit processing. In Proceedings of COLING 2012, pages 1011–1028, Mumbai, India, December. The COLING 2012 Organizing Committee.

Hatori, J., Matsuzaki, T., Miyao, Y., and Tsujii, J. (2012). Incremental joint approach to word segmentation, pos tagging, and dependency parsing in chinese. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1045–1053, Jeju Island, Korea, July. Association for Computational Linguistics.

Hellwig, O. and Nehrdich, S. (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. Association for Computational Linguistics.

Hellwig, O., (2010-2016). DCS - The Digital Corpus of Sanskrit. Berlin.

Hellwig, O. (2015a). ind. senz–ocr software for hindi, marathi, tamil, and sanskrit.

Hellwig, O. (2015b). Morphological disambiguation of classical sanskrit. In Cerstin Mahlow et al., editors, Systems and Frameworks for Computational Morphology, pages 41–59, Cham. Springer International Publishing.

Hellwig, O. (2016). Improving the morphological analysis of classical Sanskrit. In Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016), pages 142–151.

Huet, G. and Goyal, P. (2013). Design of a lean interface for Sanskrit corpus annotation. In Proceedings of ICON 2013, the 10th International Conference on NLP, pages 177–186.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20,3:331–378.

Kiparsky, P. (1994). Paninian linguistics. *The Encyclopedia of Language and Linguistics*, 6:2918–2923.

Krishna, A., Satuluri, P. K., and Goyal, P. (2017). A dataset for sanskrit word segmentation. In Proceedings of the

Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature, pages 105–114, Vancouver, Canada, August. Association for Computational Linguistics.

Krishna, A., Santra, B., Bandaru, S. P., Sahu, G., Sharma, V. D., Satuluri, P., and Goyal, P. (2018). Free as in free word order: An energy based model for word segmentation and morphological tagging in sanskrit. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2550–2561. Association for Computational Linguistics.

Krishna, A. (2019). Addressing Language Specific Characteristics for Data-Driven Modelling of Lexical, Syntactic and Prosodic Tasks in Sanskrit. Ph.D. thesis, Kharagpur, West Bengal, India.

Kudo, T., Yamamoto, K., and Matsumoto, Y. (2004). Applying conditional random fields to japanese morphological analysis. In Dekang Lin et al., editors, Proceedings of EMNLP 2004, pages 230–237, Barcelona, Spain, July. Association for Computational Linguistics.

Kudo, T. (2006). Mecab: Yet another part-of-speech and morphological analyzer. Retrieved october 9, 2019 from `https://taku910.github.io/mecab/`.

Kulkarni, A. and Ramakrishnamacharyulu, K. (2013). Parsing sanskrit texts: Some relation specific issues. In Proceedings of the 5th International Sanskrit Computational Linguistics Symposium. DK Printworld (P) Ltd.

Lowe, J. J. (2015). The syntax of sanskrit compounds. *Language*, 91(3):e71–e115.

McKinney, W. (2010). Data structures for statistical computing in python. In Stéfan van der Walt et al., editors, Proceedings of the 9th Python in Science Conference, pages 51 – 56.

More, A., Çetinoğlu, Ö., Çöltekin, Ç., Habash, N., Sagot, B., Seddah, D., Taji, D., and Tsarfaty, R. (2018). Conllul: Universal morphological lattices for universal dependency parsing. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018). European Language Resource Association.

More, A., Seker, A., Basmova, V., and Tsarfaty, R. (2019). Joint transition-based models for morpho-syntactic parsing: Parsing strategies for MRLs and a case study from modern Hebrew. *Transactions of the Association for Computational Linguistics*, 7:33–48, March.

Reddy, V., Krishna, A., Sharma, V., Gupta, P., R, V. M., and Goyal, P. (2018). Building a Word Segmenter for Sanskrit Overnight. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, May 7-12, 2018. European Language Resources Association (ELRA).

Sapir, E. (2004). Language: An introduction to the study of speech.

Scharf, P. M. and Hyman, M. D. (2011). Linguistic issues in encoding sanskrit. *The Sanskrit Library*.

Scharf, P. M. (2009). Modeling pāinian grammar. In Gérard Huet, Amba Kulkarni and Peter Scharf (Eds.), *Sanskrit Computational Linguistics: First and Second International Symposia Rocquencourt, France, October 29-31, 2007 Providence, RI, USA, May 15-17, 2008 Revised Selected and Invited Papers*. Berlin, Heidelberg:Springer Berlin Heidelberg, pp. 95–126.

Seeker, W. and Çetinoğlu, Ö. (2015). A graph-based lattice dependency parser for joint morphological segmentation and syntactic analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373.

Shivaswamy, P. and Joachims, T. (2012). Online structured prediction via coactive learning. In Proceedings of the 29th International Coference on International Conference on Machine Learning, ICML'12, pages 59–66, Edinburgh, Scotland. Omnipress.

Sims, A. D. (2015). Productivity, defectiveness, and syncretism. In *Inflectional Defectiveness*, Cambridge Studies in Linguistics. Cambridge University Press, p. 82–132.

Smith, N. A., Smith, D. A., and Tromble, R. W. (2005). Context-based morphological disambiguation with random fields. In Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pages 475–482, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Staal, F. (2008). Discovering the Vedas : origins, mantras, rituals, insights. Penguin Books India.

Susarla, S. and Challa, D. R. (2019). A platform for community-sourced indic knowledge processing at scale. In Proceedings of the 6th International Sanskrit Computational Linguistics Symposium, pages 68–82, IIT Kharagpur, India, 23–25 October. Association for Computational Linguistics.

Tratz, S. and Phan, N. (2018). A web-based system for crowd-in-the-loop dependency treebanking. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, May. European Language Resources Association (ELRA).