Design and analysis of a lean interface for Sanskrit corpus annotation

Pawan Goyal[1] *and Gérard Huet*[2]
[1] IIT Kharagpur, India
[2] INRIA Paris Laboratory, France

ABSTRACT

We describe an innovative computer interface designed to assist annotators in the efficient selection of segmentation solutions for proper tagging of Sanskrit corpora. The proposed solution uses a compact representation of the shared forest of all segmentations. The main idea is to represent the union of all segmentations, abstracting from the sandhi rules used, and aligning with the input sentence. We show that this representation provides an exponential saving, in both space and time.

The segmentation methodology is lexicon-directed. When the lexicon does not have full coverage of the corpus vocabulary, some chunks of the input may fail to be recognized. We designed a lexicon-acquisition facility, which remedies this incompleteness and makes the interface more robust.

This interface has been implemented, and is currently being applied to the annotation of the Sanskrit Library corpus. Evaluation over 1,500 sentences from the *Pañcatantra* text shows the effectiveness of the proposed interface on real corpus data.

Keywords: Sanskrit, text segmentation, annotation, interface

1 GENERALITIES ON SANSKRIT LINGUISTICS

Sanskrit is the primary language used as a vehicle of culture in India. Literature in Sanskrit for all fields of human endeavour has been produced continuously over the past four millennia, giving rise to an immense corpus, which is, to date, only partially digitised. It benefits

from a very sophisticated linguistic tradition stemming from the fairly complete grammar composed by Pāṇini by the fourth century B.C.E.

During the last 15 years, significant efforts have been made to develop computational linguistics for Sanskrit, and considerable progress has been achieved in providing computer assistance for Sanskrit corpus processing (Goyal *et al.* 2009; Hellwig 2009; Huet *et al.* 2009; Kulkarni and Huet 2009; Kulkarni and Shukl 2009; Scharf and Hyman 2009; Jha 2010; Kulkarni *et al.* 2010; Kumar *et al.* 2010; Goyal *et al.* 2012). Nevertheless, there does not exist at this time a complete analyser for Classical Sanskrit texts able to compute morphological tagging reliably in a completely automatic way. The main difficulty concerns segmentation, since Sanskrit is represented in writing by continuous phonetic enunciation, which demands complex processing for its analysis in separate word forms. Although complete algorithms for this segmentation preprocessing have been proposed (Huet 2005), human assistance is still needed to focus on the appropriate solution within all possible analyses.

We propose in this paper a new human-machine interface to help a professional annotator to decide quickly between all possible segmentations, in order to select a unique morphological analysis among the many possible ones. Indeed, there exist thousands of such segmentations for simple sentences, and literally billions for complex ones. Once a sufficient amount of tagged corpus data has been made available using such semi-automated annotation tools, it is hoped that it will be possible to use it to train a fully automated parser using statistical methods.

A preliminary version of this paper was presented at the ICON conference in Hyderabad in December 2013 (Huet and Goyal 2013). The novel aspects and contributions of this paper with respect to the previous version are: a) it is an extended version of the conference paper, with a detailed explanation of the segmentation methodology, related work and illustrative examples, b) we conduct a thorough evaluation of the proposed system with respect to robustness as well as convergence time taken, in practice, on real corpus data with 1,500 sentences, and c) we propose a module for error recovery and lexical acquisition, which makes the system much more usable when dealing with a corpus that is error-prone, or contains words that are not present in the lexicon used by the system.

The paper is organized as follows. Section 2 discusses related work on the problem of word segmentation. Section 3 gives the necessary formalisms required for segmentation analysis of Sanskrit text. The concept of aligned segmentations and the graphical display of the interface, which are the central themes of this paper, are detailed in Sections 4, 5 and 6. The use of the proposed interface as a tagging tool is discussed in Section 7, using a complete walk-through example. We present the evaluation of the proposed interface using 1,500 sentences from real corpus data in Section 8. An experimental segmenter for lexical acquisition is described in Section 9. Section 10 concludes the paper.

2 RELATED WORK

The task of word segmentation is a necessary initial step for processing those natural languages where word boundaries are not maintained in the written text. A lot of prior work concerns word segmentation for Chinese text. The two dominant models for Chinese word segmentation are 'word-based' and 'character-based' (Sun 2010). Word-based methods read the input sentences from left to right, predicting whether the current piece of continuous characters is a word token. Once a word is found, they move on and search for the next word. The methods vary in terms of the strategies used for word prediction and disambiguation, if there are multiple possibilities. For instance, the maximum matching approach (Chen and Liu 1992) chooses the longest word for disambiguation, while prediction is based on a dictionary. Recently, machine-learning methods have been employed to solve these problems. Zhang and Clark (2007) used a linear model with an average perceptron algorithm where, given an input sequence of characters c, the model finds a segmentation $\hat{w}$ such that

$$\hat{w} = \max_{w \in GEN(c)} (\alpha \cdot \phi(c, w)),$$

where ϕ is a feature map, α is the parameter vector, which is learnt via training, and $GEN(c)$ enumerates the set of segmentation candidates for the character sequence c.

Character-based approaches, on the other hand, attempt to assign labels to the characters in the sequence, indicating whether a character c_i is a single character word (S), or the beginning (B), middle (I)

or end (E) of a multi-character word, thus treating this as a sequence labelling problem. Word tokens are inferred, based on the character classes. Several models, such as Conditional Random Fields (CRFs), have been used for this task (Tseng 2005).

Various approaches have since been proposed to combine word-based and character-based methods (Sun *et al.* 2009). For instance, Wang *et al.* (2014) recently proposed a method based on dual decomposition (Rush *et al.* 2010) to combine these approaches in an efficient framework.

The problem of word segmentation in Sanskrit, however, is more difficult than in languages such as Chinese, where the words are combined without any euphonic assimilation at the boundary. The next section describes in further detail the problem of word segmentation for Sanskrit text.

3 SEGMENTATION ANALYSIS FOR SANSKRIT TEXT

We shall now formalize the word segmentation problem in Sanskrit written text at various levels of abstraction. Sanskrit may be written in all Indian scripts, most usually in the *devanāgarī* script used by languages of North India such as Hindi, but such syllabic representation is awkward for morpho-phonetic computations. It is preferable to translate it into a list of phonemes, with one-to-one translation. We assume the standard set of 50 phonemes, already known from the time of Pāṇini. Such low-level representation issues are discussed at length in Scharf and Hyman (2009) and Huet (2009).

In Sanskrit, where oral tradition dominated the sphere of learning and an advanced discipline of phonetics explicitly described euphonic assimilation, the phonetic transformations at the juncture of successive words, well known by the term *sandhi*, are represented in writing. Assimilation obscures word boundaries in speech, and these word boundaries are correspondingly eliminated in writing as well. For example, *vasati* ‘dwells’ *atra* ‘here’ becomes *vasatyatra* in continuous speech. Some euphonic changes, like this one, can be separated in alphabetic Roman transcription despite the sound alteration, viz. *vasaty atra*. Other assimilation changes, however, preclude word separation, even in alphabetic transcription, because the final sound of the preceding word and the initial sound of the following word merge

into a single sound. Thus *vidyā* 'knowledge' *āpyate* 'is attained' becomes *vidyāpyate*; the single sound *ā* belongs to both words. This phenomenon has been well recognized and formally analysed since antiquity (Pāṇini gave a complete axiomatisation of sandhi in terms of string rewriting in his 4th century B.C. treatise Aṣṭādhyāyī). The most difficult task in parsing a Sanskrit sentence is determining the word boundaries. Solutions to this problem have valuable ramifications for speech analysis, where a similar problem is encountered in virtually all languages.

We assume that the reader is familiar with the use of finite-state methods for morpho-phonemic computations, as explained in standard references such as Roche and Schabes (1997), Kaplan and Kay (1994) and Beesley and Karttunen (2003). We also assume some familiarity with the lexicon-driven Sanskrit segmenter of Huet (2005), from which we extract the following definitions.

Definitions. A *lexical juncture system* on a finite alphabet Σ is composed of a finite set of words $L \subseteq \Sigma^*$ and a finite set R of rewrite rules of the form $[x]u|v \rightarrow w$, with $x, v, w \in \Sigma^*$ and $u \in \Sigma^+$. Note that in the Kaplan and Kay notation, the rule we write $[x]u|v \rightarrow w$ would be written as $u|v \rightarrow w/x\_$.[1]

The word $s \in \Sigma^*$ is said to be a *solution* to the system (L, R) iff there exists a sequence $\langle z_1, \sigma_1 \rangle; \ldots; \langle z_p, \sigma_p \rangle$ with $z_j \in L$ and $\sigma_j = [x_j]u_j|v_j \rightarrow w_j \in R$ 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 $v_0 = \epsilon$, and finally $s = s_1 \ldots s_p$ with $s_j = y_j x_j w_j$ for $(1 \leq j \leq p)$. ϵ denotes the empty word. We also say that such a sequence is an *analysis* of the solution word s.

In this formalization, Σ is the set of phonemes, R is the set of sandhi rules, and L is the vocabulary as a set of lexical items. As a first approximation, one may think of L as the lexicon of inflected words. In Section 7, we shall partition L according to lexical sorts, some of which are morphemes such as stems and affixes, in order to segment compound words by the same method as explained here for sentence

[1] This algorithm assumes that the segmenter induces the segment boundaries from a generative lexicon of permitted inflected forms. Another method would guess arbitrary segment boundaries with rules $uv \rightarrow w/x\_$, and attempt morphological analysis of the segments, but this is less efficient. Some rules could also use the v part as right context, when it is unchanged. Many variations exist.

segmentation into words. This extension is necessary to keep L finite, in view of the fact that nominal compounding in Sanskrit is productive to an arbitrary depth. But all the notions defined here will apply easily to this refinement, which allows us to keep notations simple. We shall also assume the system (L,R) to be *non-overlapping*, as defined in Huet (2005). This assumption is met in classical Sanskrit, except for a small number of uni-phonemic morphemes, which are amenable to the general treatment, modulo the introduction of so-called *phantom phonemes*, as explained in Huet (2006); Goyal and Huet (2013).

Note that the sandhi problem is expressed in a symmetric way. Going from $z_1|z_2|\ldots|z_n| \in (L\cdot|)^*$ to $s \in \Sigma^*$ generates a correct phonemic sentence s with word forms $z_1, z_2, \ldots, z_n$, using the sandhi transformations. Whereas going the other way means analysing the sentence s as a possible phonemic stream, using words from the lexicon transformed by sandhi. It is this second problem that the Sanskrit segmenter has to solve, since sandhi, while mostly deterministic in generation, is strongly ambiguous in analysis. Below, we provide a brief summary of the solution proposed by Huet (2005). The basic data structures used by the system are *Tries*, *Decos*, and variations on applicative data structures to represent finite automata and transducers. This methodology has recently been extended to a general paradigm of relational programming, using the notion of effective Eilenberg machines (Huet and Razet 2015).

Definitions. *Tries* are tree structures that store finite sets of strings sharing initial prefixes. We assume that the alphabet of string representations is some initial segment of positive integers. Thus a string is encoded as a list of integers that will from now on be called a *word*.

Definitions. A word may be associated with a non-empty list of information of polymorphic type α, absence of information being encoded by the empty list. We shall call such associations a *decorated trie*, or *deco* for short.

To solve the sandhi problem for analysis, the inflected form tries are decorated with the rewrite rules. The algorithm proceeds in one bottom-up sweep over each inflected form trie. For every accepting node (i.e. lexicon word), at occurrence z, we collect all sandhi rules[2]

[2] The treatment of a contextual rule $[x]u|v \to w$ is similar: we check that $z = \lambda xu$, but the decorated state is now at occurrence λx. In both kinds of rules,

$\sigma : u|v \rightarrow w$ such that u is a terminal substring of z: $z = \lambda u$ for some λ. When we move up the trie, recursively building the automaton graph, we decorate the node at occurrence λ with a choice point labelled with the sandhi rule. This builds in the automaton the prediction structure for rule σ, at distance u above a matching lexicon word. At interpretation time, when we enter the state corresponding to λ, we shall consider this rule as a possible non-deterministic choice, provided the input tape contains w as an initial substring. If this is the case, we shall then move to the state of the automaton at occurrence v (before this, the program checks that all sandhi rules are plausible in the sense that occurrence v exists in the inflected trie, i.e. there are some words that start with string v). When we take this action, the automaton acts as a transducer, by writing on its output tape the pair (z, σ).

Coming back to the solution word, we may think of s as a phonetically correct utterance over vocabulary L, and its analysis $S = \langle z_1, \sigma_1 \rangle; \ldots; \langle z_p, \sigma_p \rangle$ as one of its possible segmentations. Analysis S is completely explicit, in the sense that s may be computed from S, applying sandhi rules σ_i in sequence, going from left to right. Conversely, there may be many possible segmentations S of a given utterance s, typically thousands for a moderately long sentence, although it is proven in Huet (2005) that they are always finite in number. We write $Segs(s)$ for the set of segmentations of s. The algorithm described in Huet (2005) shows how to enumerate the complete set $Segs(s)$ from a given input string s. In view of its possibly enormous size, attempts have been made, e.g. Huet (2007), to filter out non-sensible segmentations by a semantic analysis in the manner of dependency grammars. This method works well for simple sentences, but is not sufficient for more complex sentences, particularly in the presence of ellipses and other anaphoric or discourse operators where dependencies are context-sensitive. Furthermore, the set $Segs(s)$ is not easily amenable to sharing, and as a consequence the segmentation-cum-tagging Web service of the Sanskrit Heritage site[3] has not been of practical use so

---

the choice point is put at the ancestor of z at distance u. This suggests as implementation to compute at the accepting node z a stack of choice points arranged by the lengths of their left component u. Furthermore, once the matching is done, the context x may be dropped when stacking a contextual rule, since it is no longer needed.

[3] `http://sanskrit.inria.fr/`

far on real corpus data, since it tended to generate very long Web pages, even to the point of choking the server. Wading through such long lists of segmentations was very tedious and error-prone. The new interface described in the present paper completely solves this problem. We shall now explain its main concepts.

4 ALIGNED SEGMENTATIONS

The key idea behind the new interface is to represent an abstraction of the union of all segmentation decompositions, realigned on the input utterance. This new representation is now amenable to sharing, and may thus be represented very compactly on one computer screen.

Definition. We consider a sandhi analysis S as above, generalized to allow empty sequences. It may be defined inductively, as being either empty or of the form $S = \langle z_1, \sigma_1 \rangle ; S'$, with S' a similar sequence. Let n be a natural number. We define the *alignment* of S *with offset* n, noted as $S \hookrightarrow n$, as a set of pairs of *aligned segments* of the form (k, z), with $k \in \mathbb{N}$ and $z \in L$, as follows. If S is the empty sequence, then $S \hookrightarrow n = \emptyset$. Otherwise, let $S = \langle z, \sigma \rangle ; S'$ with $\sigma = [x]u|v \rightarrow w$. We define $S \hookrightarrow n = \{(n, z)\} \cup S' \hookrightarrow n'$, where $n' = n + |z| + |w| - (|u| + |v|)$.

If S is a segmentation analysis of utterance s, we define its corresponding *aligned segment collection* as the set of aligned segments $\overline{S} = S \hookrightarrow 0$. Note that in this new notion we leave aside the precise sandhi rules used in the analysis S, keeping only the tabulation information that allows us to present its set of segments aligned with the original input s.

Let $\mathscr{S}$ be a set of segmentation analyses of utterance s. We define the *tabulated display* of $\mathscr{S}$, noted $D(\mathscr{S})$, as the set of aligned segments obtained as the union of all its corresponding aligned segment collections:

$$D(\mathscr{S}) = \bigcup_{S \in \mathscr{S}} \overline{S}$$

We say that an aligned segment (k, z) is *relevant* to a segmentation analysis S iff $(k, z) \in \overline{S}$. Let $\mathscr{S}$ be a non-empty set of segmentation analyses of some utterance s, and $(k, z) \in D(\mathscr{S})$. We define the *restriction* of $\mathscr{S}$ to (k, z), noted $\mathscr{S} \downarrow (k, z)$, as the set of all segmentation analyses in $\mathscr{S}$ to which (k, z) is relevant:

$$\mathscr{S} \downarrow (k, z) = \{S \in \mathscr{S} \mid (k, z) \in \overline{S}\}$$

We obtain of course $\mathscr{S} \downarrow (k, z) \subseteq \mathscr{S}$.

Fact 1. $(k,z) \in D(\mathscr{S}) \Rightarrow \mathscr{S} \downarrow (k,z) \neq \emptyset$.
Proof. Trivial compactness property of union.

Let $\mathscr{S}$ be a non-empty set of segmentation analyses of some utterance s, and $(k,z) \in D(\mathscr{S})$. We say that (k,z) is *critical* in $D(\mathscr{S})$ iff it is not relevant to some $S' \in \mathscr{S}$. This implies that

$$|D(\mathscr{S} \downarrow (k,z))| < |D(\mathscr{S})|$$

Thus, selecting a critical segment in the interface will effectively reduce the search space. In practice, it will reduce it by half or more, and convergence will be ensured in $\log(N)$ steps, where N is the total number of segmentation solutions. Let us now give a sufficient condition for criticality.

Let (k,z) and (k',z') be two distinct aligned segments in some tabulated display $D(\mathscr{S})$. We say that (k,z) and (k',z') *conflict* if $k \leq k' < k+|z|-1$ or $k' \leq k < k'+|z'|-1$.
Fact 2. Let (k,z) and (k',z') conflict in $D(\mathscr{S})$. They are both critical, as they are mutually exclusive – no segmentation may contain both.
Proof. By inspection of sandhi rules, we may check that every rule $[x]u|v \rightarrow w$ is such that $|u|+|v| \leq |w|+1$. Thus any overlap of a segment with its successor in any segmentation is at most of length one. Since every segment is of length at least one, overlap of a segment with some other segment in the same segmentation solution may be at most of length one. Let (k',z') be an aligned segment of $D(\mathscr{S})$ conflicting with (k,z). No segmentation analysis to which (k',z') is relevant may belong to $\mathscr{S} \downarrow (k,z)$, and thus $(k',z') \notin D(\mathscr{S} \downarrow (k,z))$.

Note that the conflicting condition is sufficient to show that two segments may not appear in a common segmentation solution, but that this is not a necessary condition, even for contiguous segments. The interest of this notion is that it is easy to check visually, whereas the necessary and sufficient criterion is not, since sandhi rules are not shown.

We now state a fact which may not be true of all lexical juncture systems, but is verified for Sanskrit sandhi, as we shall argue in Section 7.3.
Fact 3. If $D(\mathscr{S})$ has no critical aligned segment, $\mathscr{S}$ is a singleton.

5 A GRAPHICAL INTERFACE

Let s be the utterance under consideration. Initially, we compute the set $\mathscr{S} = Segs(s)$ of all its possible segmentations, and we display $D(\mathscr{S})$,

where every aligned segment (k, z) is represented as the segment z displayed with an offset of k spaces from the left margin. When two aligned segments overlap, we represent them in separate rows of the display. We sort all segments, so that longer segments are listed above shorter ones. Each segment is displayed either with a blue check sign, if it does not conflict with any other segment, or else with two signs, a green check sign to select the segment, and a red cross sign to discard it. These green check and red cross signs are mouse-sensitive; they trigger as call-back the segmentation routine that will compute all segmentation analyses consistent with this particular choice, that is, for which all aligned segments currently selected with green check signs are present, and those segments discarded with red cross signs are absent. If s is segmentable at all, $\mathscr{S}$ is non-empty, and so is $D(\mathscr{S})$. At any point in the computation, the current display $D(\mathscr{S})$ represents the union of a non-empty set $\mathscr{S}$ of segmentations of s, by repeated application of Fact 1. Consequently, selecting or discarding a segment can never fail.

Furthermore, if the user selects or discards a critical segment, there is visible progress, since all conflicting segments vanish when a segment is selected, while any segment vanishes when discarded. This corresponds to the case where it conflicts with some other segment, which is easy to see in the visual display, since it covers a column that is strictly inside the conflicting segment.

When a segment is selected using the green check sign, both the check and cross signs are replaced by a single blue check sign, which is mouse-insensitive, thus making the segment inert for the rest of the interaction. On the other hand, if a segment is discarded using the red cross sign, it vanishes and in the particular case where it conflicts only with one other segment, the other segment will become inert. Note that the user cannot select a non-critical segment, since these are presented with blue check signs, which are not mouse-sensitive. When there are no more critical segments, we have reached a unique segmentation solution, consistent with Fact 3.

Several other actions, besides the selection of a segment, are possible at any moment. Firstly, users may undo the previous selections, up to an arbitrary depth. Secondly, they may revert to the old interface, which gives a linear listing of all segmentations consistent with the segments currently selected. A counter indicates how many

distinct segmentations remain. Users may also opt to use the semantic pruning mechanism to provide machine assistance, for potentially faster convergence. Finally, it is possible to send the remaining set of segmentations to the more complete dependency parser under development at the University of Hyderabad (Kulkarni *et al.* 2010; Kulkarni and Ramakrishnamacharyulu 2013).

A complexity analysis of the interface is presented in the Appendix. From experimental evidence, it has been observed that the number of solutions grows exponentially with the length of the utterance and the bound $O(C^n)$ has actually been reached for the real corpus (see Figure 10). For instance, the following sentence, excerpted from the *Vikramorvaśī* play by Kālidāsa, has 6,967,296,000 ($\approx 2^{32}$) segmentations. The sentence has 240 phonemes, and the desired solution has 40 segments. This sentence can be managed by our interface in 17 clicks, so the convergence is quite fast.
yā tapasviśeṣapariśaṅkitasya sukumāram praharaṇam mahendrasya pratyādeśaḥ rūpagarvitāyāḥ śriyaḥ alaṃkāraḥ svargasya sānaḥ priyasakhī urvaśī kuberabhavanāt pratinivartamānā samāpattidṛṣṭenakeśinā dānavenacitralekhādvitīyā bandigrāhaṃ gṛhītā. 'Our dear friend *Ūrvaśī*, who is the youthful weapon of *Mahendra*, the one fearful of the power of extra-ordinary penance, who is an overshadower of *Śrī*, who is proud of her beauty, and who is an ornament of heaven, was taken captive, together with *Citralekhā*, by the demon *Keśī*, who had appeared by chance, while she was returning from the house of *Kubera*.' (English translation by Brendan Gillon)

More example cases will be presented in Section 8.

6 GRAPHICAL RENDERING OF THE DISPLAY

Figure 1 presents the graphical rendering presented by our system for the following sentence:
satyaṃbrūyātpriyaṃbrūyānnabrūyātsatyamapriyaṃpriyaṃcanānṛtambrūyādeṣadharmaḥsanātanaḥ.
This is the well-known saying (*subhāṣitam*): 'One should tell the truth, one should say kind words; one should neither tell harsh truths, nor flattering lies; this is a rule for all times.'

As indicated in the display, this diagram summarizes 120 distinct segmentations. The colour code used for the segments indicates var-

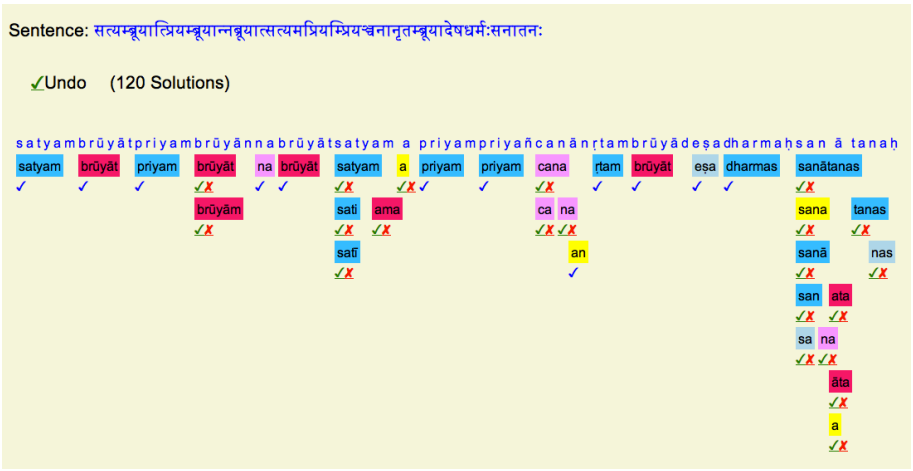

Figure 1: Initial display of the aligned segments for the sentence *satyaṃbrūyātpriyaṃbrūyānnabrūyātsatyamapriyaṃpriyaṃcanānṛtambrūyādeṣa-dharmaḥsanātanaḥ*

ious lexical categories, e.g., blue for substantives, red for finite verb forms, purple for adverbs, pale blue for pronouns and yellow for compounds.

The main notion behind the interface is that of the display $D(\mathscr{S})$ for a consistent set of segmentations $\mathscr{S}$. Initially, we take $\mathscr{S} = Segs(s)$, and we progressively select aligned segments $(k_1, z_1), \ldots, (k_n, z_n)$. The only data kept in memory are the initial sentence s, and the stack of choices $C_n = (k_1, z_1), \ldots, (k_n, z_n)$. The interface interaction is implemented as a CGI coroutine, which receives arguments s and C_n in its invoking URL. The server recomputes the sequence of all segmentations $Segs(s)$ at every step, keeping only those consistent with the stack of choices C_n, sorted by alignments into a sorted list of checkpoints. The display of all consistent segmentations is stored in an array 'display' of size $|s|$. The display value at index i is the list of all segments z such that (i, z) is an aligned segment of some segmentation solution consistent (i.e. not conflicting) with all the checkpoints C_n. This test is easy, since C_n is sorted. One may think of the display as a shared representation of $D(\mathscr{S})$, for $\mathscr{S}$, the set of segmentation solutions consistent with the current stack of choices. Actually the array 'display' may be thought of as a hashcoding array for the set $D(\mathscr{S})$, with the hashcode of an aligned segment (k, z) being its alignment k in the input string.

What is crucial for the efficient sharing of the tree of all segmentations as a directed acyclic graph (DAG) is the abstraction of sandhi rules. Indeed, our methodology is reminiscent of parsers based on tabulation methods, which use such dynamic programming methods (Earley 1983; Tomita 1985; Billot and Lang 1989; Stolcke 1995).

Implementation of 'Undo' is trivial, since it consists in calling the interface with the same stack of choices minus the last choice.

Note the simplicity of this implementation: at every step, all the information is recomputed with the standard segmenter but, since the technology is very fast, this is not noticeable to the user as the reaction seems instantaneous (at least on a localhost server).

Presenting the tabulated display of the aligned segmentations as an HTML page was not entirely trivial. The segmentation analysis gives us all possible segments, appearing at various offsets. First, for an arbitrary offset k_i, the number of segments may be quite large. Also, the length $|z_i|$ of the largest segment (k_i, z_i) at offset k_i might be such that it conflicts with the aligned segments at the next offset k_{i+1}. Since the objective was to have a compact display, keeping the alignment intact, the problem of where to fit the aligned segments at offset k_{i+1} remains, in such a case, once the HTML display has been populated with the segments at offset k_i. The second issue is related to the fact that, while the maximum size of the display array is fixed as the length of the utterance ($|s|$), the size of an aligned segment (k_i, z_i) is $|z_i|$, a variable depending on the segment z_i. Thus, the problem is to show the aligned segment as a single entity.

Now, a simplistic implementation to keep the alignment intact would have been to list all the segments corresponding to the offset k_{i+1}, starting from the next row after all the segments at offset k_i have been enumerated. This would obviously not lead to a compact display. Similarly, a very simple implementation to handle variable-sized segments would be to define an array of $|s|$ columns and display each solution (k_i, z_i) in $|z_i|$ columns, starting from the k_i^{th} column. The problem with this approach is that the display of a word does not appear continuous here. Also, depending upon the transliteration scheme used, some phonemes would require more space than others, so the row length will be variable. And the segment z_i cannot be treated as a single HTML entity in this case, which is a requirement for user-friendly display

of morphological tags, as well as for the callbacks, initiating user interaction.

To alleviate these problems, we sorted the segments at each offset according to length.[4] Thus, the longer segments appear at the top. Now, while filling the segment (k_{i+1}, z_{i+1}) at offset k_{i+1}, we search for the first row from the top where the last filled segment does not conflict with (k_{i+1}, z_{i+1}), and fill this segment in that row. This gives a much more compact display.

Similarly, to handle the second issue, instead of using $|z_i|$ columns for an aligned segment (k_i, z_i), we used the HTML 'colspan' attribute to use variable width columns in a row. Thus, an aligned (k_i, z_i) is displayed, using a $|z_i|$ width column at offset k_i. This allows continuous display of a segment, as well as treating it as a single HTML entity.

7 LEXICAL CATEGORIES AND TAGGING

7.1 *Dealing with lemmatized segments*

Since our method is lexicon-directed, our candidate forms are morphologically generated, and may be kept along with their lemmas. Furthermore, we may restrict our segmenter to recognize only morphologically correct sequences, according to a regular grammar expressing morphological constraints. This refinement is also necessary because the sandhi relation after preverbs (*upasarga*) is different from the external sandhi between words or compound components. This grammar is compiled into the state-transition graph of a finite automaton/transducer, which expresses the control of our lexical scanner in the usual manner. The states of this automaton, called *phases*, correspond to the lexical categories associated with colours in the interface. We may refine the above formalization to this new situation easily, replacing the notion of aligned segment (k, z) by the finer notion of *aligned lemmatized segment* $(k, (l, z))$, where l is the lemmatization of segment z.

We can go back to the example sentence in Section 6, for which the initial display summarizing 120 distinct segmentations is presented in Figure 1. At the right side of the diagram, one sees the long

[4] Note that this sorting is prioritized from left to right, as this is the most natural order for reading Sanskrit text.

segment *sanātanas* (‘eternal’) and below it several choices of smaller words that are obviously overgenerated items. Clicking on the green sign under the blue segment *sanātanas* removes all this noise, and the number of potential solutions drops to 12, generating the display given in Figure 2 – note the blue unlinked check sign indicating the previously selected segment.

Similarly, one immediately notices the segment *satyam* (‘truth’), together with conflicting noisy alternatives. Similarly, for *cana* (‘and not’), these two selections will leave us with only one choice between segments *brūyāt* and *brūyām* (two forms of root *brū* ‘to say’ in the optative mood of the present active voice in the singular number, respectively in the 3rd and 1st person). By obvious symmetry with its other occurrences in the sentence, *brūyāt* must be chosen, obtaining the correct segmentation in a total of 4 easy clicks, shown in Figure 3. At this point, one may click on the explicit button “Unique Solution”, where fine tuning of the final morphological parameters, such as ambiguities of gender of substantival forms, may be effected through a final user interface, shown in Figure 4. This last stage is necessary, because our lemmas label a given form with a multi-tag, factoring out all values of morphological parameters usable to generate this form. The user can select the appropriate options to produce the final unambiguous tagging of the sentence as a list of lemmas, where segments are hyperlinks to the digital lexicon, as shown in Figure 5.

This page may be stored, and the next sentence may then be read from the corpus input stream, in order to progressively annotate the digital library.

Sometimes it is useful for annotators to see the lemmatization of a segment in order to make a decision with more information than merely its lexical category (indicated by the colour code). This facility

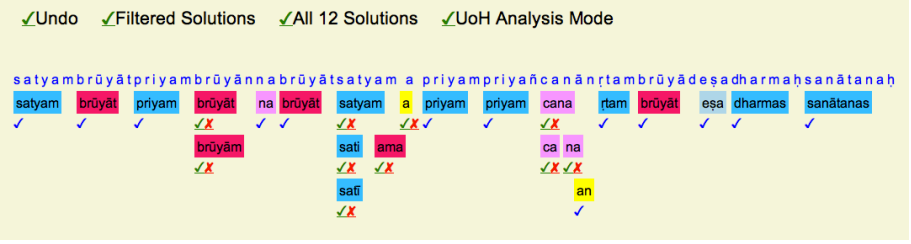

Figure 2: Aligned segments after selection of segment *sanātanas*

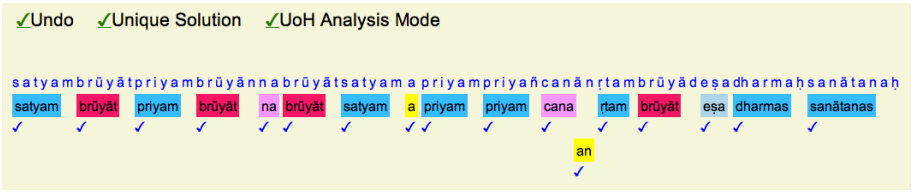

Figure 3: Aligned segments after 4 clicks

Figure 4: The interface for selecting unique tags from multi-tags

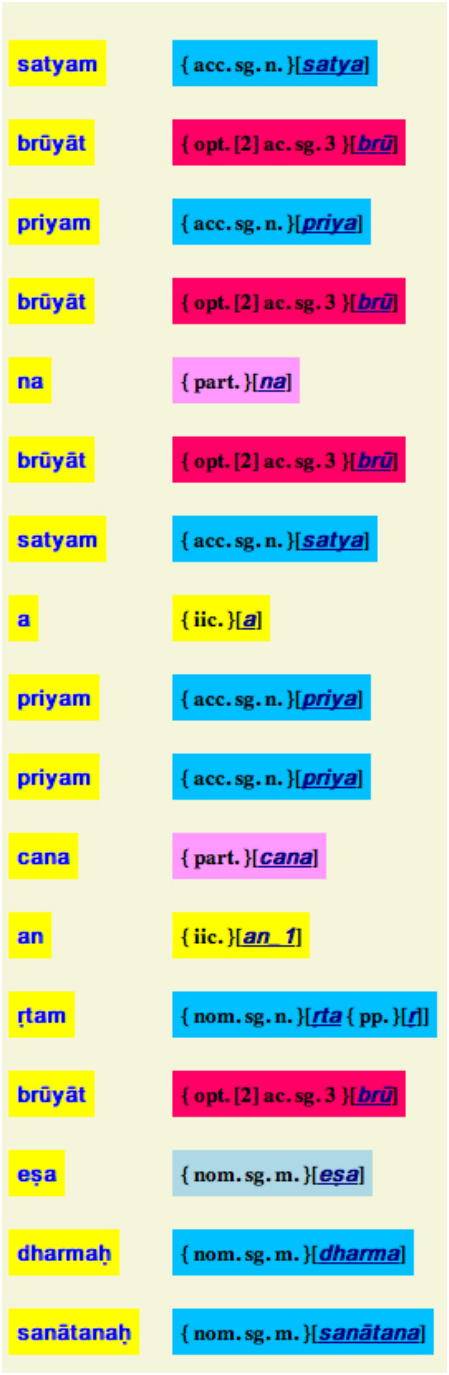

Figure 5:
Final tagging

is available in the interface: every segment is mouse-sensitive, and clicking on it yields its lemma, as shown in Figure 6 for the segment *brūyāt*.

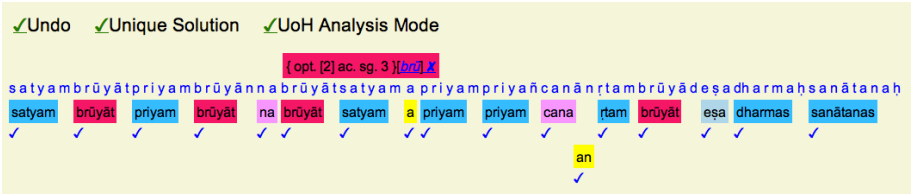

Figure 6: Asking for the lemma of the segment *brūyāt*

Note that, in this lemma, the root *brū* is itself mouse-sensitive; it is a hyperlink to its lexical entry, allowing access to its meaning. We provide two aligned digital lexicons, our original Sanskrit-to-French Heritage dictionary, and also the more complete classical Sanskrit-to-English Monier-Williams (MW) dictionary (Monier-Williams *et al.* 1899).[5] Thus annotators have all available information at their disposal at any point, but with minimal cluttering of the workspace.

It should be noted that this interface is not only easy to use, it is actually fun to play with. It may be thought of as some kind of electronic game.

7.2 *Rationale for using the cross signs*

The cross signs presented for conflicting segments are used to discard a particular segment. However, it may be argued that this result may more efficiently be achieved by selecting the correct segment. While this would be the appropriate in majority of cases, there are a few instances where one would need to use a cross sign to select the appropriate solution. Figure 7 describes the possible segmentations for the utterance, *ihehi*, which can be analysed either as *iha* + *ā* + *ihi* or as *iha* + *ihi*. The interface presents the segments *iha* and *ihi* with blue check signs, indicating that these do not conflict with any other segment, but the segment *ā* is presented with a green check and a red cross sign. If we had used only a green check sign, it would not have been possible for the annotator to select the solution *iha* + *ihi*,

[5] The protocol for the non-trivial task of mutually linking these lexical resources has been discussed in Goyal *et al.* (2012)

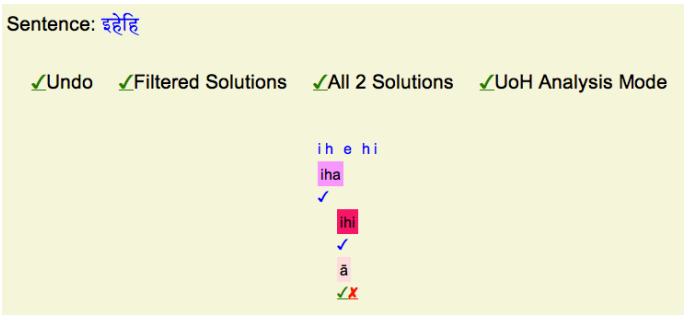

Figure 7:
The aligned segmentations for *ihehi*.

since there would have been no opportunity to discard the *ā* segment. With this facility, the annotator is free to choose either of these two analyses.[6]

7.3 *Justifying Fact 3*

In the example just shown, we assumed implicitly that when no more choices were available to the user, there was only one segmentation solution left, and we could then proceed to the final disambiguation of the remaining multi-tags of this unique solution. This assumption is precisely what we called Fact 3 above, and that we now restate:

Fact 3. If $D(\mathscr{S})$ has no critical aligned segment, $\mathscr{S}$ is a singleton.

Proof. Assume that $D(\mathscr{S})$ has no critical aligned segment. In other words, all the segments are marked with a blue mark, indicating that they belong to all remaining solutions. Thus, all remaining solutions have the same segments. We shall need to prove that all the aligned segments are strictly ordered within one unique solution. Consider any two remaining segments (k,z) and (k',z') where, without loss of generality, we may assume $k \leq k'$. If $k < k'$, the z segment must precede the z' one. Now let $k = k'$. It is not the case that both $|z| > 1$ and $|z'| > 1$, since the two segments would conflict with each other. Assume without loss of generality $|z| = 1$. If $|z'| > 1$, the z segment must precede the z' one. We are left to consider the case where $|z| = |z'| = 1$. The only relevant mono-phonemic segments in classical Sanskrit are the privative prefix *a*, forming so-called *nañ-tatpuruṣa* compounds, and the

[6] Another possible way to achieve this would have been to use null segments and allow the annotators to choose between the null segment or the other possibility. We, however, prefer to use the cross sign, since it also helps a reader to reduce the number of possibilities by discarding some nonsensical combinations.

preposition *ā*, used as prefix (*upasarga*) to final (*tiṅanta*) and propositional (*kṛdanta*) verbal forms.[7] We thus only have to consider the proper ordering of co-aligned *a* and *ā* segments. The privative particle *a* can prefix only consonant-initial nouns, since it alternates with the form *an* for vowel-initial ones. The preposition *ā* is assumed not to be iterated, which would be redundant. Thus, the only possible ordering is that an *ā* segment could precede an *a* segment (but we do not know of even one concrete example). This explanation justifies Fact 3 in the case of classical Sanskrit.

7.4 *Robustness*

The interface is remarkably robust for realistic sentences, as shown in the example in Section 5. Figure 8 shows the initial display of our interface, where the sentence from the *Vikramorvaśī* play by Kālidāsa, as mentioned in Section 5, is processed by the Sanskrit reader. The interface shows all the 6,967,296,000 possible solutions in a compact display. The display presents various choice points to the user, and is manageable in 17 clicks. A full evaluation of the interface, for robustness as well as convergence analysis, is presented in the next section.

8 EVALUATION

To evaluate the robustness of the proposed system, we used a dataset consisting of 1,500 sentences from *Pañcatantra*. These sentences were annotated, using a software-assisted human interface for morphological tagging, built on top of the Sanskrit Heritage Reader (Goyal *et al.* 2012). The annotators were allowed to give their own annotations, when the correct segmentation did not appear in the system. The length distribution of the sentences used in this study is shown in Figure 9. Clearly, many of the sentences contained more than 90 characters.

To study its robustness, we checked whether the sets of segmentation analyses for these sentences contained the segmentations identified by the annotators. When these sentences were given as input to the system, the system gave a summary page in *each of the cases*.

[7] Vedic Sanskrit offers additional difficulties, with autonomous prepositions and the mono-phonemic interjection *u*.

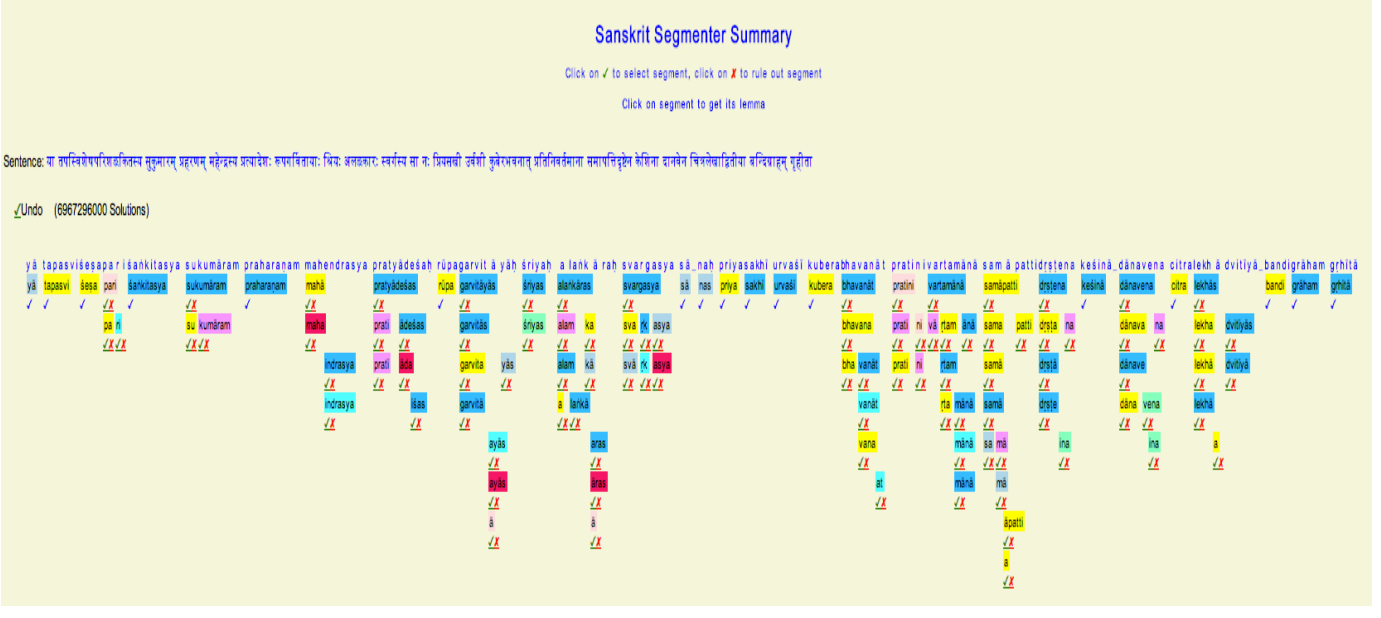

Figure 8: Checking the interface for a long sentence

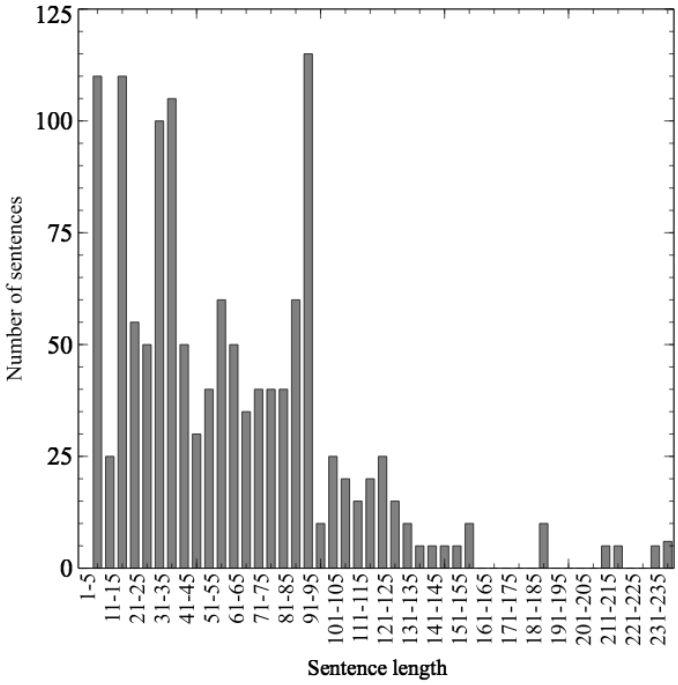

Figure 9: Distribution of length for the sentences used in the evaluation

Figure 10 plots the log of the number of solutions identified by the system, with respect to the length of the sentence. The plot is truncated at 125 characters both for the sake of visibility and also because few sentences exceed that limit. We can clearly see that the number of solutions increased exponentially with the length of the sentence.

On further analysis, we found that in 1,092 out of 1,500 sentences, all the segmentations from the gold standard were present in the set of segmentation analyses, returned by the summary interface of the system. On analysing the rest of the cases, we found that, in 59 cases, the annotated sentence did not match the input sentence, and a few changes had been introduced by the annotators. We therefore studied the performance of the system on the remaining 1,441 sentences. First, we measured the recall of the system by identifying how many of the words in the segmentation were also present in the summary interface. We measured both micro- and macro-averaged recall. As per the standard definition, for macro-averaged recall, we computed recall for each of the sentences and then took an average for all 1,441 values, one for each sentence. For micro-averaged recall, we computed the

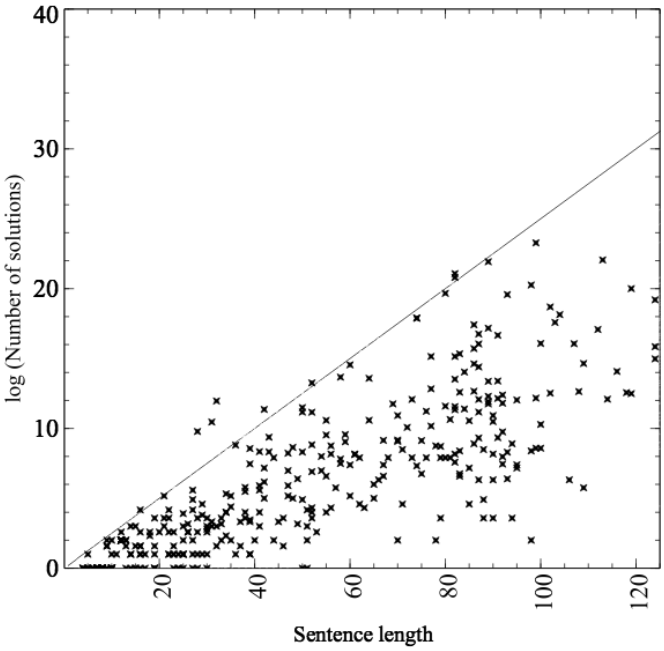

Figure 10:
Scatter plot showing the distribution of the number of solutions (log) with respect to the length of the sentence

fraction of words present overall. These values were found to be high: 0.96 and 0.97 respectively.

For 349 cases, one or more words in the gold-standard segmentations could not be mapped to the segmentations returned by the summary interface. A further analysis revealed that, in 204 of these cases, the system could not recognize a word from the sentence, mostly because of the incompleteness of the lexicon used by the Sanskrit Heritage system. This problem can be solved by supplementing the lexicon with new words specific to the particular corpus under consideration. In the next section, we describe how the current system helps annotators to do this in an interactive manner. For most of the other cases, the main issue was that the original sentence contained a quote, or that a punctuation mark occurred in the middle of the sentence, e.g., the sentence,[8] *tatas tayā, manorathānām apy agamyam, iti matvā, tathā, iti pratipannam* was pre-processed and the following was the input to the system: *tatas tayā manorathānām apy agamyam iti matvā tathā iti prati-*

[8] And she assented, for she thought: "It is a thing beyond my fondest aspirations." (English translation)

pannam, resulting in the words *tathā iti* being separated by a space in 'sandhied' mode.[9] This was not recognized by the system, as it denotes the *pada-pāṭha* ('unsandhied') form and not the sandhied form. In 'sandhied' mode, the correct input would have been *tatheti*, as *tathā iti* leads to other interpretations, such as *tathās + iti*, *tathau + iti*, etc. In future, we might be able to auto-detect this, along the lines of spell-correction.

In many such cases, the system could not make use of sentence breaks and punctuation information, which were removed during pre-processing. The system has to be adapted to allow such information in the input, to be able to make adjustments during segmentation.

Further, to empirically evaluate the convergence time to get the unique solution out of all the possible segmentations returned by the system, we took a sample of 10 sentences of length ≥ 100, and the annotators were asked to use the summary interface to come up with the unique solution. We noted down the actual time taken, as well as the number of clicks used by the evaluators. The details are provided in Table 1 below. The length of the sentences varied between 113 and 224, and the total solutions were as high as 3,736,212,480. However, in all cases, at most 19 clicks were required to achieve the unique

Table 1: Empirical evaluation of the convergence time for 10 different sentences

S. No.	Sentence length	Total solutions	Number of clicks	Time (in sec.)
1	150	22,394,880	14	59.2
2	115	4,368	6	28.2
3	156	19,051,200	17	56.3
4	224	248,832,000	17	73.6
5	149	18,662,400	14	42.9
6	149	3,736,212,480	19	78.7
7	113	2,880	8	32.3
8	122	9,216	8	17.0
9	122	17,600	10	36.4
10	169	167,215,104	17	65.8

[9] This is one of the parameters of the system. The user can choose the mode 'sandhied' to read a sentence that is not segmented and the mode 'unsandhied' to read text that has already been sandhi analysed (*pada-pāṭha* form).

solution, and the maximum time taken was 78.7 seconds, which is quite fast, as well as practical.

The segmentation method is lexicon-directed. Thus, for any aligned segment (k_i, z_i) to appear in the segmentation solution, the segment z_i must belong to the vocabulary L. It will thus be incomplete, if the generative lexicon does not completely cover the vocabulary of the targeted corpus. In the next section, we will describe how our interface is robust enough to handle the cases when a chunk (part of the utterance s, which is segmentable, independent of the rest of the utterance) is not recognized by the system.

9 PARTIAL SEGMENTATION, ERROR RECOVERY, LEXICAL ACQUISITION

In general, given an utterance s, a chunk may remain 'unanalysed' or 'ill-analysed'. The case of 'unanalysed' chunks might occur due to one of the following reasons:

- The utterance s contains an invalid chunk z_i, not allowed by the grammar, or
- s contains a segment (chunk) z_i, which is a valid segment, but does not appear in the vocabulary L, due to the incompleteness of the lexicon.[10]

A chunk may remain ill-analysed if the desired solution does not appear in the segmentation returned by the system. This mostly occurs because of the incompleteness of the lexicon.

In order to deal with this incompleteness, and make our interface robust, we extended it in such a way that it will report the unanalysed chunks of input, and allow for their correction. This facility has been provided by adding a supplementary phase to the lexer, allowing any phonemic string. Thus, when the system is unable to recognize a segment z_i in utterance s at offset k_i, this unanalysed segment is displayed in grey along with a spade sign. This spade sign triggers as callback another CGI routine called 'user-aid', initiating an interaction loop with the user.

[10] Note however that our lexicon is 'generative' to a certain extent: most participles (*kṛdantas*) are systematically generated from root entries, and compounds are analysed, and thus do not need to be explicitly listed in the lexicon.

For the 'ill-analysed' chunks, since there is at least a partial solution, no explicit link is provided to the 'user-aid' CGI. Instead, if the user decides that the analysis presented by the system is not correct for some particular chunk, clicking on the chunk will provide access to the 'user-aid' CGI for the given chunk. This routine provides the following options:

Edit and resubmit the sentence. If the user has entered a wrong sentence *s* (for instance due to misspelling), this option allows the user to edit the sentence *s* and submit it to the system for re-analysis.

Edit and resubmit the chunk. This option allows the user to edit only the wrong chunk in *s* and does not disturb the rest of the sentence. The user can edit the chunk and the system will show the analysis corresponding to the modified chunk, keeping the segmentation solution of the other chunks intact.

Show partial solution without this chunk. This option appears only when there are at least two chunks in the sentence. This option allows the user to see the partial solution without using that chunk.

Select among possible lemmatizations. This module tries to guess the possible lemmatizations (analyses) of a segment from the 'Unknown' phase. This module is developed using finite state methods and will be discussed in Section 9.1 below.

Enter your own lemmatization. If users feel that none of the suggested lemmatizations are correct, this option allows them to enter the lemmatizations of their choice. This module will be discussed in Section 9.2 below.

9.1 *Experimental Stemmer*

We have implemented an experimental stemmer, in order to attempt semi-automatic lexicon acquisition, at least for substantive stems. This is a very difficult problem, in the presence of retroflexion rules by internal morphology. This progressive assimilation of the retroflex articulatory feature operates on a non-bounded left context of the rule application, and thus cannot be directly modelled as an invertible regular transduction. Fortunately, retroflexion rules do not cross word boundaries, and thus do not pollute external sandhi.

The experimental module for guessing the possible lemmatizations for an 'unanalysed' chunk is built using the suffix segmentation

rules, learnt from the database of inflected forms, available with the Heritage lexicon. To give an example of the rules learnt, consider the following entry in the database of inflected noun forms:

rāmas nom.sg.m. [rāma] 'Rama, name of a person'

The entry has three different parts, the inflected form *rāmas*, the stem corresponding to this form *rāma* and the morphological information of this inflected entry 'nom.sg.m.'. This entry is used to learn the following rule:

$$x.a \xrightarrow{nom.sg.m.} x.as \tag{1}$$

where $x.a$ denotes any phonetic string ending in the phoneme 'a'. Similarly, for the entry,

takṣan loc.sg.m. [takṣṇi] 'carpenter'

The rule learnt would be

$$x.an \xrightarrow{loc.sg.m.} x.ṇi \tag{2}$$

Note that, in both these cases, the context (right context, or ending) is chosen based on the following criteria:

- The context should not be empty. This condition was used so that the rule would not cause an over-generation. Thus we will not learn the rule $x.\phi \xrightarrow{nom.sg.m.} x.s$ in the first case where ϕ denotes a null context.
- The minimum possible context should be used to describe the rule. This condition was used to avoid the segmenter failing because of having too long a context. Thus we will not learn the rule $x.ma \xrightarrow{nom.sg.m.} x.mas$ in the first case, because then it would not allow us to recognize that the word *mohanas* is a declined form of the stem *mohana* 'Mohana, name of a person' because the context *nas* would not match the one used in the rule (*mas*).

Now, since the database also contains very special rules, which might be applicable to only a few stems, a simple probabilistic model is used to filter these rules. The first filter is based on the frequency count of a certain rule, that is, how many times this rule is encountered while declining the nominal forms in the lexicon. Rule 1, above, is used 5079 times, while rule 2 is used only 13 times.

The next filter is based on the conditional probability of a stem and morphological analysis being associated with a given suffix. Thus, a rule is selected only if the probability of the stem and morphologi-

cal analysis given the suffix is greater than the threshold. For rules 1 and 2 discussed above, these probability values were found to be 0.96 and 0.05 respectively. For rule 2, this probability was low because, given the ending *ṇi*, the stem ending in *n* with the same morphology is more likely (probability of 0.63). An example of one such entry in the database is: *dīrghasūtrin loc.sg.m. [dīrghasūtriṇi]* 'spinning a long yarn, procastinating'.

Thus, two different thresholds are used, frequency count and probability. The criteria for selecting these thresholds involved a trade-off. A low value for these thresholds would allow too many unnecessary solutions for a given segment. A very high threshold, on the other hand, might not be able to provide the desired solution. Thus, these values were tuned on a corpus,[11] resulting in optimal values of 3 for frequency and 0.02 for probability.

Once these rules were learnt from the database, they were fed into a finite-state transducer, which could then be used to guess all the possible stems along with the morphological analysis for a previously unanalysed form. All the possible lemmatizations produced by the transducer are displayed to the user. The interaction loop for lexicon acquisition is discussed in the next section.

9.2 *Lexicon acquisition*

For a segment in the 'Unknown' phase, various lemmatizations are proposed by the transducer. They are presented to the user accompanied by radio buttons. These radio buttons allow the user to select among the various suggested lemmatizations. It should be noted that one of the objectives of this module is to acquire the stems that appear in the corpus but are not available in the Heritage lexicon. To assist the user, we search for each suggested stem in the Monier-Williams (MW) dictionary, which is one of the most complete lexicons for Sanskrit. If a stem appears in the MW, the stem is displayed with a hyperlink to the online MW dictionary, and the radio button corresponding to this entry is preset. This is based on the intuition that, among all the possible choices, any choice that is already present in a more complete lexicon is more likely to be correct, and will, in any case, be verified

[11] We collected examples of unanalysed segments from the Bhagavad Gītā text. These examples were used to tune the thresholds.

by the user. If the user selects any of these suggestions, the base entry and gender information are saved in a 'cache' database.

If the users cannot find the desired solution among the suggested lemmatizations, they are allowed to enter their own lemmatization. A text area is provided for the user to enter the stem, with various select boxes, to be completed with morphological information, such as gender, case and number. Once the user submits this information, the base entry and gender information is saved in the 'cache' database.

To illustrate this procedure, we input the following sentence from *Pañcatantra* into the Sanskrit reader: *ye punar ātmīyāḥ śṛgālā āsan te sarve 'py ardhacandraṃ dattvā niḥsāritāḥ*. 'But to all the jackals, his own kindred, he administered a cuffing, and drove them away.'

Figure 11 shows the aligned segments as returned by the system. The system does not present any analysis for the segment *ātmīyāḥ* (his own kindred), which is displayed in grey, along with a spade sign.

Once the annotator clicks on the spade sign, it opens the 'user-aid' CGI routine. Various options presented to the annotator by this routine are shown in Figure 12. In this particular case, the segment *ātmīyāḥ* is a valid segment, which remains unanalysed because the stem *ātmīya* is not present in the lexicon. Thus, we will focus on the option 'Select among possible lemmatizations'. The annotator is presented with various possible analyses but the specific analysis with the stem *ātmīya* present in MW has been shown with a hyperlink. The annotator can select the radio button corresponding to the first analysis in the

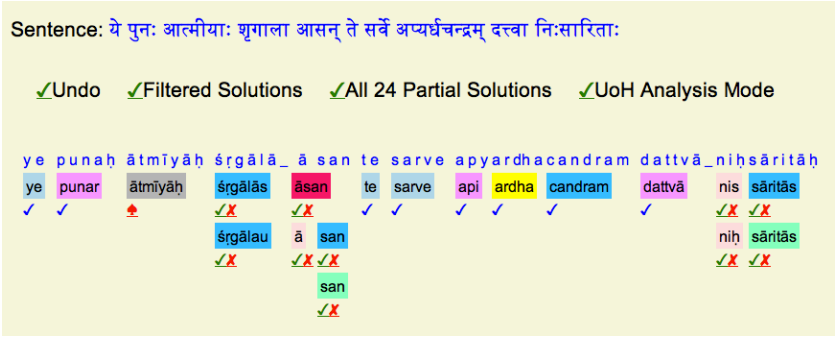

Figure 11: The partial segmentations for the sentence *ye punar ātmīyāḥ śṛgālā āsan te sarve 'py ardhacandraṃ dattvā niḥsāritāḥ* with the segment *ātmīyāḥ* remaining unanalysed

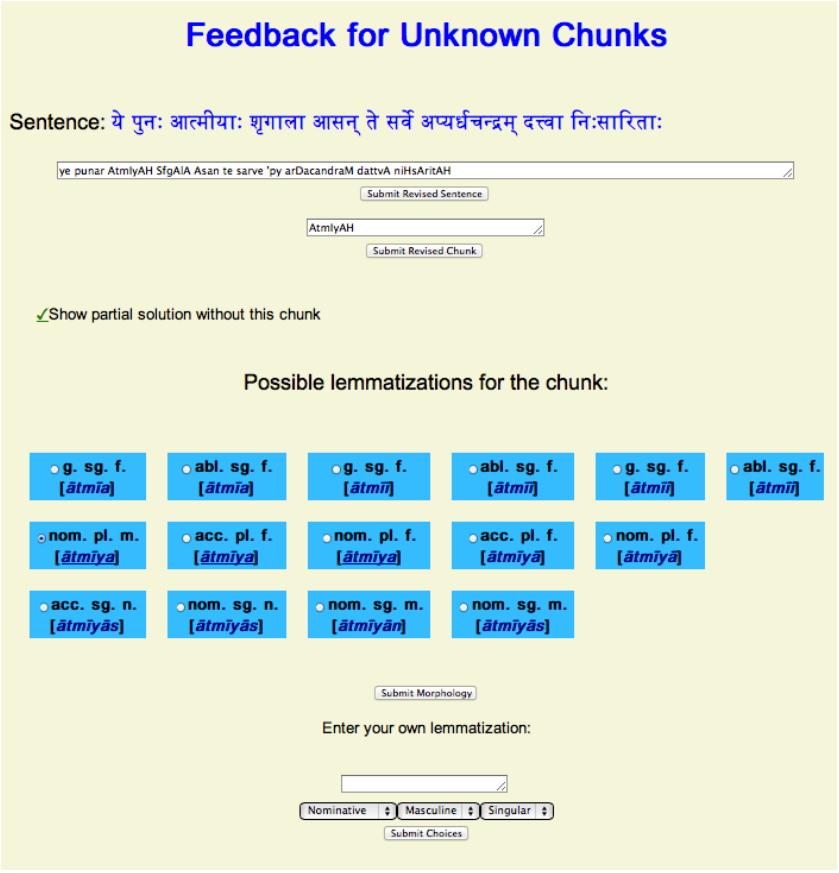

Figure 12: Options provided to the annotator for the unanalysed chunk *ātmīyāḥ*

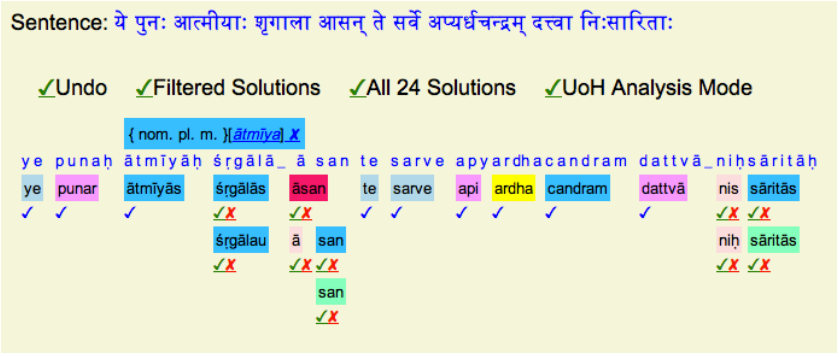

Figure 13: Revised interface for the annotator with *ātmīyāḥ* analysed as chosen, using the options shown in Figure 12

second row (nom. pl. m. [*ātmīya*]) and click on the button 'Submit Morphology'.

This information provided by the annotator is stored in the 'cache' database. The morphological generator is used to generate all the forms corresponding to the stems stored in this database. This cache database augments the lexicon L, and thus enables the system to recognize a segment that was previously unanalysed or ill-analysed. Figure 13 shows the revised interface that is presented to the annotator, with the segment *ātmīya* analysed as chosen. Now the annotator can complete the tagging by going through the normal process, as already described in detail.

It is to be noted that, once this stem is processed to augment the lexicon L, the system can recognize all other inflected forms corresponding to this base stem as well.[12] This feature is particularly useful for annotators working on a specific corpus, since an unanalysed stem is likely to appear in that corpus again, possibly as a different utterance in another morphological context. The information about the selected stem and gender is stored in a local file on the annotator's workstation, which may be passed on to the lexicon manager for lexicon acquisition.

9.3 *Evaluating the experimental stemmer*

To evaluate the experimental stemmer, we used the 53 nominal forms that were not recognized by the system. These 53 words were passed to the 'user-aid' CGI routine. Among the suggestions provided by the system, we selected the particular lemma that corresponds to the stem in the Monier-Williams dictionary, which would have given that nominal form. On manual verification, we found that, in 52 out of 53 cases, the lemma matched the one provided by the annotators. This confirms that this experimental stemmer can be used very effectively by the annotators to deal with words that are unknown to the system.

[12] All the paradigms for generating the nominal and verbal forms are already available in the system. Thus, given a new nominal stem as input, the system can generate all its inflected forms, which are added to the database. At the time of analysis, all these forms are therefore recognizable.

10 CONCLUSION

We have presented a new interface for interactive segmentation-cum-tagging of Sanskrit sentences. This technology is not limited to Sanskrit. It can be adapted for interactive feedback, with a segmenter, tagger or parser, where sentences are presented as a finite collection of sequences of annotated word forms (lemmas). It may also operate at the generative morphology level, where words are presented as a combination of morphemes.

This interface enables a human annotator to visualise a sentence as a sequence of words, readable in one compact hypertext page. Word forms are vertically aligned with the original input. This allows the sharing of lemmas, and avoids cluttering the visual display with redundant information. Segments at a given offset are sorted by length, in decreasing order, which permits easy selection, with a heuristic of maximum overlap of segments with the input sentence. This heuristic, which tends to minimize the number of segments, is very often correct. Small word forms or morphemes, which agglutinate by chance into larger chunks of the input, get relegated as noise to the bottom of the display screen.

Fast recomputation of solutions respecting selection or rejection of a given segment achieves an exponential convergence rate. Even for long sentences admitting billions of solutions, the effect of these selections is instantaneous. Selection mistakes may be fixed rapidly using the undo facility. Morphological information is hidden in order not to clutter the screen, since appropriate use of colours for lexical categories usually facilitates the right decision. In case of doubt, the annotator may click on any puzzling segment and instantly obtain its full lemmatization, including lexicon access, if required, to check the meaning.

The main concept behind the data structure containing the display information is dynamic programming, i.e. sharing a tree structure as a directed acyclic graph, a standard technique in tabulated parsers. The originality of our approach is that the tree structure is not the forest of parse trees, but the union of all possible segmentation solutions, from which sandhi justification has been erased. This representation allows exponential savings, both in space (the displayed graph) and in time (the number of disambiguation operations).

The main ideas of this interface have been reused to summarize all possible dependencies between word forms in the dependency parser developed at the Sanskrit Studies Department of the University of Hyderabad.[13] This parser may be accessed as a second pass of our segmenter, leading to a smooth combination of the two processes – the user switches seamlessly between tagging and parsing (Huet and Kulkarni 2014). When to call the parser is actually an interesting trade-off. If we call it too early, it will just choke under the enormous number of possible taggings. On the other hand, if we use our manual interface until we have produced a single set of tags, we lose many of the benefits of automation, since the dependency analysis would discard many inconsistent word combinations.

We have presented a novel technique for lexicon acquisition during corpus tagging by annotators, which makes our interface robust to lexicon incompleteness, but also to corpus mistakes and to non-standard enunciations (non-Paninian forms, Prakrit,[14] onomatopoeia, foreign words, etc.). The current module is developed only for nominal forms and needs to be extended to handle verbal forms as well. Another limitation of this module is that the system would only be able to guess a stem if the unanalysed chunk contains only one word. Handling cases where the unanalysed chunk contains more than one word is the next logical goal for our project.

Our interface has been tested successfully by the Sanskrit Library team[15] for the annotation of a variety of classical Sanskrit texts (Scharf *et al.* 2015).

APPENDIX: COMPLEXITY ANALYSIS

The convergence of the selection via the interface is very fast. Since the method is dichotomic, it converges on average in $\log(N)$ steps, where N is the total number of segmentation solutions. Indeed, when the input may be split as $s = s_1 \cdot s_2$, with s_1 and s_2 independently segmentable, with respectively n_1 and n_2 segmentations, presented with displays of sizes respectively d_1 and d_2, the global display has a size of $d_1 + d_2$ for

[13] `http://sanskrit.uohyd.ac.in/scl/`

[14] By the term 'Prakrit', we mean Middle Indo-Aryan languages such as Pāli.

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

a total of $n_1 \times n_2$ segmentations. This interface thus gives an exponential improvement over the recursive dove-tailing of the segmentation process. In any case, the number of selections will be smaller than the number of words of the intended segmentation, i.e. of the order of the length of the sentence divided by the average length of a word. In practice, convergence is very fast.

Theorem. Let $\mathscr{S}$ be the set of segmentation analyses of some utterance s of length n. $|\mathscr{S}|$ is of asymptotic order $O(C^n)$, whereas $|D(\mathscr{S})|$ is of asymptotic order $O(n)$.

Proof. This theorem depends on the lexicon being used and can have, at best, an average complexity analysis. Let m be the length of an average segment of an utterance s. For our analysis, we will also assume that each segment in a valid solution has length ≥ 2.

Consider s of length n. We will try to find an upper bound on the number of segmentation solutions for this utterance. Let us consider the i^{th} phoneme of this utterance. A valid solution can have this phoneme participating in a segment of length 2, 3, ... up to m. Analysing further, a segment of length 2 can start at 2 possible offsets, $i-1$ or i. Similarly, a segment of length 3 can start at 3 possible offsets, and so on. In general, let of_j denote the number of offsets at which a segment of length j may start for the i^{th} phoneme. Then, $of_j \leq j$ for $j \in \{2, 3, \ldots, m\}$. Every such offset k for a segment of length j defines a set with aligned segments (k, z_l) such that $|z_l| = j$. Thus, for the i^{th} phoneme, an upper bound on the number N_{ss_i} of possible sets is:

$$\begin{aligned} N_{ss_i} &\leq of_2 + of_3 + \cdots + of_m \\ &\leq 2 + 3 + \cdots + m \\ &< \frac{m(m+1)}{2} \end{aligned} \tag{3}$$

For each of these N_{ss_i} sets, the possible number of segments depends on the sandhi rules R. For any segment in such a set, permutations are possible only at the first and last phonemes because of the sandhi rules applied at the junction. Let $left_w$ denote the number of possible v's, such that $u|v \rightarrow w \in R$ for an arbitrary u. Similarly, let $right_w$ denote the number of possible u's, such that $u|v \rightarrow w \in R$ for an arbitrary v. Now let *maxleft* be the maximum of all such $left_w$ and *maxright* be the maximum of all such $right_w$. Thus, such a set can contain at most $|ss_i| = (maxleft \times maxright)$ segments. Then, the maximum number of

segments N_i that the i^{th} phoneme can participate in is:

$$N_i \leq |ss_i| \cdot N_{ss_i} \tag{4}$$

Now that we have the maximum number of possible segments for the phoneme at position i, we can use this to obtain an upper bound on the number of segments $|\mathscr{S}|$ for the utterance s. We will use the fact that the set $|\mathscr{S}|$ will be a subset of all the possible segments in which phonemes at various positions can participate. Thus

$$\begin{aligned} |\mathscr{S}| &\leq N_1 \times N_2 \times \cdots \times N_n \\ &= (|ss_i|)^n \cdot {N_{ss_i}}^n \\ &= \left(C \cdot \frac{m(m+1)}{2}\right)^n \end{aligned} \tag{5}$$

Similarly, an upper bound on the number of segments in the tabulated display is the sum of all possible segments at various positions. Thus

$$\begin{aligned} |D(\mathscr{S})| &\leq N_1 + N_2 + \cdots + N_n \\ &= \left(C \cdot \frac{m(m+1)}{2}\right) \cdot n \end{aligned} \tag{6}$$

Hence, it follows from Equations 5 and 6 that $|\mathscr{S}|$ is of asymptotic order $O(C^n)$ at worst, whereas $|D(\mathscr{S})|$ is of asymptotic order $O(n)$.

REFERENCES

Kenneth R. Beesley and Lauri Karttunen (2003), *Finite-state morphology: Xerox tools and techniques*, CSLI Publications, The University of Chicago Press.

Sylvie Billot and Bernard Lang (1989), The structure of shared forests in ambiguous parsing, in *Proceedings of the 27th annual meeting on Association for Computational Linguistics*, ACL '89, pp. 143–151, Association for Computational Linguistics, Stroudsburg, PA, USA, doi:10.3115/981623.981641.

Keh-Jiann Chen and Shing-Huan Liu (1992), Word identification for Mandarin Chinese sentences, in *Proceedings of the 14th conference on Computational linguistics-Volume 1*, pp. 101–107, Association for Computational Linguistics.

Jay Earley (1983), An efficient context-free parsing algorithm (reprint), *Communications of the ACM - Special 25th Anniversary Issue*, 26(1):57–61.

Pawan GOYAL, Vipul ARORA, and Laxmidhar BEHERA (2009), Analysis of Sanskrit text: Parsing and semantic relations, in Gérard HUET, Amba KULKARNI, and Peter SCHARF, editors, *Sanskrit Computational Linguistics 1 & 2*, pp. 200–218, Springer-Verlag LNAI 5402.

Pawan GOYAL and Gérard HUET (2013), Completeness analysis of a Sanskrit reader, in Malhar KULKARNI, editor, *Recent Researches in Sanskrit Computational Linguistics (Proceedings, 5th International Symposium on Sanskrit Computational Linguistics)*, pp. 130–171, D.K. Printworld.

Pawan GOYAL, Gérard HUET, Amba KULKARNI, Peter SCHARF, and Ralph BUNKER (2012), A distributed platform for Sanskrit processing, in *COLING*, pp. 1011–1028.

Oliver HELLWIG (2009), SanskritTagger, a stochastic lexical and POS tagger for Sanskrit, in Gérard HUET, Amba KULKARNI, and Peter SCHARF, editors, *Sanskrit Computational Linguistics 1 & 2*, pp. 266–277, Springer-Verlag LNAI 5402.

Gérard HUET (2005), A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger, *Journal of Functional Programming*, 15,4:573–614, `http://yquem.inria.fr/~huet/PUBLIC/tagger.pdf`.

Gérard HUET (2006), Lexicon-directed segmentation and tagging of Sanskrit, in Bertil TIKKANEN and Heinrich HETTRICH, editors, *Themes and Tasks in Old and Middle Indo-Aryan Linguistics*, pp. 307–325, Motilal Banarsidass.

Gérard HUET (2007), Shallow syntax analysis in Sanskrit guided by semantic nets constraints, in *Proceedings of the 2006 International Workshop on Research Issues in Digital Libraries*, pp. 6:1–6:10, ACM, New York, NY, USA, doi:http://doi.acm.org/10.1145/1364742.1364750, `http://yquem.inria.fr/~huet/PUBLIC/IWRIDL.pdf`.

Gérard HUET (2009), Formal structure of Sanskrit text: Requirements analysis for a mechanical Sanskrit processor, in Gérard HUET, Amba KULKARNI, and Peter SCHARF, editors, *Sanskrit Computational Linguistics. First and Second International Symposia Rocquencourt, France, October 29-31, 2007 Providence, RI, USA, May 15-17, 2008*, pp. 162–199, Springer.

Gérard HUET and Pawan GOYAL (2013), Design of a lean interface for Sanskrit corpus annotation, in *Proceedings of ICON 2013, the 10th International Conference on NLP*, pp. 177–186.

Gérard HUET and Amba KULKARNI (2014), Sanskrit linguistics web services, in *COLING (Demo)*, pp. 48–51.

Gérard HUET, Amba KULKARNI, and Peter SCHARF, editors (2009), *Sanskrit computational linguistics 1 & 2*, Springer-Verlag LNAI 5402.

Gérard HUET and Benoît RAZET (2015), Computing with relational machines, *Mathematical Structures in Computer Science*, FirstView:1–20, ISSN 1469-8072, doi:10.1017/S0960129515000390, `http://journals.cambridge.org/article_S0960129515000390`.

Girish Nath JHA, editor (2010), *Sanskrit computational linguistics 4*, Springer-Verlag LNAI 6465.

Ronald M. KAPLAN and Martin KAY (1994), Regular models of phonological rule systems, *Computational Linguistics*, 20,3:331–378.

Amba KULKARNI and Gérard HUET, editors (2009), *Sanskrit computational linguistics 3*, Springer-Verlag LNAI 5406.

Amba KULKARNI, Sheetal POKAR, and Devanand SHUKL (2010), Designing a constraint based parser for Sanskrit, in Girish N. JHA, editor, *Proceedings of the 4th International Sanskrit Computational Linguistics Symposium*, pp. 70–90, Springer-Verlag LNAI 6465.

Amba KULKARNI and K. V. RAMAKRISHNAMACHARYULU (2013), Parsing Sanskrit texts: Some relation specific issues, in Malhar KULKARNI, editor, *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium*, pp. 191–212, D. K. Printworld(P) Ltd.

Amba KULKARNI and Devanand SHUKL (2009), Sanskrit morphological analyser: Some issues, *Indian Linguistics*, 70(1-4):169–177.

Anil KUMAR, Vipul MITTAL, and Amba KULKARNI (2010), Sanskrit compound processor, in Girish N. JHA, editor, *Proceedings of the 4th International Sanskrit Computational Linguistics Symposium*, pp. 57–69, Springer-Verlag LNAI 6465.

Monier MONIER-WILLIAMS, Ernst LEUMANN, and Carl CAPPELLER (1899), *A Sanskrit-English Dictionary: Etymological And philologically arranged with special reference to cognate Indo-European languages*, Oxford, The Clarendon Press, `http://www.sanskrit-lexicon.uni-koeln.de/scans/csldoc/dictionaries/mw.html`.

Emmanuel ROCHE and Yves SCHABES (1997), *Finite-State Language Processing*, MIT Press.

Alexander M. RUSH, David SONTAG, Michael COLLINS, and Tommi JAAKKOLA (2010), On dual decomposition and linear programming relaxations for natural language processing, in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1–11, Association for Computational Linguistics.

Peter SCHARF, Anuja AJOTIKAR, Sampada SAVARDEKAR, and Pawan GOYAL (2015), Distinctive features of poetic syntax preliminary results, *Sanskrit syntax*, pp. 305–324.

Peter SCHARF and Malcolm HYMAN (2009), *Linguistic issues in encoding Sanskrit*, Motilal Banarsidass.

Andreas STOLCKE (1995), An efficient probabilistic context-free parsing algorithm that computes prefix probabilities, *Computational Linguistics*, 21(2):165–201.

Weiwei SUN (2010), Word-based and character-based word segmentation models: Comparison and combination, in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 1211–1219, Association for Computational Linguistics.

Xu SUN, Yaozhong ZHANG, Takuya MATSUZAKI, Yoshimasa TSURUOKA, and Jun'ichi TSUJII (2009), A discriminative latent variable Chinese segmenter with hybrid word/character information, in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 56–64, Association for Computational Linguistics.

Masaru TOMITA (1985), *Efficient parsing for natural language: A fast algorithm for practical systems*, The Springer International Series in Engineering and Computer Science - Volume 8, Springer.

Huihsin TSENG (2005), A conditional random field word segmenter, in *Fourth SIGHAN Workshop on Chinese Language Processing*.

Mengqiu WANG, Rob VOIGT, and Christopher D. MANNING (2014), Two knives cut better than one: Chinese word segmentation with dual decomposition, in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014), Baltimore, MD*.

Yue ZHANG and Stephen CLARK (2007), Chinese segmentation with a word-based perceptron algorithm, in *Annual Meeting of the Association for Computational Linguistics*, volume 45, p. 840.