A Graph-Based Framework for Structured Prediction Tasks in Sanskrit

Amrith Krishna*
Department of Computer Science
and Technology
University of Cambridge
ak2329@cam.ac.uk

Bishal Santra
Department of Computer Science
and Engineering
Indian Institute of Technology
Kharagpur
bsantraigi@gmail.com

Ashim Gupta†
School of Computing
University of Utah
ashim@cs.utah.edu

Pavankumar Satuluri
School of Linguistics & Literary Studies
Chinmaya Vishwavidyapeeth
pavankumarsatuluri@gmail.com

Pawan Goyal
Department of Computer Science
and Engineering
Indian Institute of Technology
Kharagpur
pawang@cse.iitkgp.ac.in

* Work done while at Indian Institute of Technology Kharagpur. Email: ak2329@cam.ac.uk
† Work done while at Indian Institute of Technology Kharagpur.

Submission received: 23 July 2019; revised version received: 25 August 2020; accepted for publication: 3 October 2020.

https://doi.org/10.1162/COLI_a_00390

We propose a framework using energy-based models for multiple structured prediction tasks in Sanskrit. Ours is an arc-factored model, similar to the graph-based parsing approaches, and we consider the tasks of word segmentation, morphological parsing, dependency parsing, syntactic linearization, and prosodification, a "prosody-level" task we introduce in this work. Ours is a search-based structured prediction framework, which expects a graph as input, where relevant linguistic information is encoded in the nodes, and the edges are then used to indicate the association between these nodes. Typically, the state-of-the-art models for morphosyntactic tasks in morphologically rich languages still rely on hand-crafted features for their performance. But here, we automate the learning of the feature function. The feature function so learned, along with the search space we construct, encode relevant linguistic information for the tasks we consider. This enables us to substantially reduce the training data requirements to as low as 10%, as compared to the data requirements for the neural state-of-the-art models. Our experiments in Czech and Sanskrit show the language-agnostic nature of the framework, where we train highly competitive models for both the languages. Moreover, our framework enables us to incorporate language-specific constraints to prune the search space and to filter the candidates during inference. We obtain significant improvements in morphosyntactic tasks for Sanskrit by incorporating language-specific constraints into the model. In all the tasks we discuss for Sanskrit, we either achieve state-of-the-art results or ours is the only data-driven solution for those tasks.

1. Introduction

Sentence constructions in morphologically rich languages (MRLs), such as Sanskrit, generally rely on morphological markers to encode the grammatical information (Tsarfaty, Sima'an, and Scha 2009). This makes Sanskrit a relatively free word order language (Staal 1967; Gillon and Shaer 2005). In fact, the same sentence follows a different word order when written as a verse, as compared to the word order in prose (Tubb and Boose 2007). However, the sentence will still maintain the same syntactic analysis, irrespective of its varying word orders (Scharf et al. 2015; Gillon and Shaer 2005). Recently, Krishna et al. (2018) have shown that approaches for non-sequential processing of Sanskrit sentences result in better system performance even for low-level tasks such as word-segmentation and morphological parsing. In this work, we extend the energy-based model (EBM) for joint modeling of word segmentation and morphological parsing proposed by Krishna et al. (2018) into a general graph-based parsing framework for multiple structured prediction tasks in Sanskrit. We extend the framework to include two downstream syntax-level tasks, dependency parsing and syntactic linearization. We also introduce the prosodification task where a bag of words is taken as input, and a verse sequence, where the sequence follows a metrical pattern, is predicted. In prosodification, only the prosody-level information, and no morphosyntactic information, about a given input is used. Figure 1 shows the hierarchy of the tasks. The challenges arising from the computational treatment of Sanskrit fall somewhere between speech recognition and the analysis of written text (Huet 2005). The written representation in Sanskrit is actually a phonemic stream (Huet 2005). The word boundaries in Sanskrit are not always explicitly marked, and are often obscured because of phonetic transformations at word boundaries. The fusional language has rich morphology, and suffers from ambiguity because of syncretisms and homonymy. Further, the "case" information from the morphological markers is

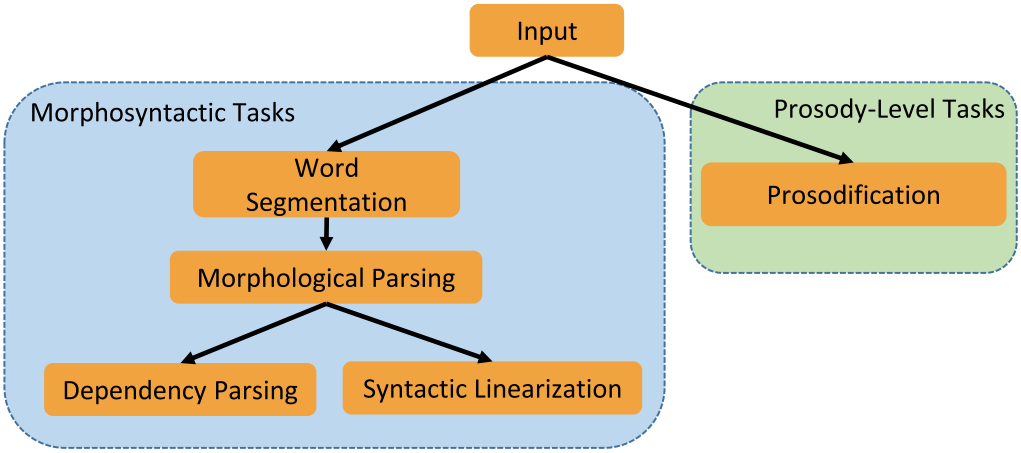

Figure 1
Hierarchy of the tasks.

crucial for identifying the syntactic roles between words in their dependency analysis (Kiparsky and Staal 1969). The case to dependency relation mapping is often a many-to-many mapping, and contextual information is required to resolve the ambiguity in such cases. Even a small eight-character string, *nagarāṇi*[1], can create ambiguities because of multiple possible word segmentation, morphological, and syntactic analyzes. *Nagarāṇi* can either be segmented as a two-word sequence *na garāṇi* (no poisons) or be treated as an inflected form of *nagara* (town) (Krishna, Satuluri, and Goyal 2017). Assuming the latter is correct, *nagarāṇi* is a plural form of the neuter gender stem *nagara*, which can either be in nominative, vocative, or accusative case. Assuming the inflection to be in nominative-case, this information enables the nominal to form one of the two possible syntactic relations with the main verb in the sentence, namely, *kartā* (subject) or *karma* (object)[2], in the syntactic analysis of the sentence. The cyclic dependency between morphological and syntax-level tasks is well known (Tsarfaty 2006), and these tasks are often solved jointly (More et al. 2019). Similarly, the potential error propagation from word segmentation to its downstream tasks in pipeline models is also well established for multiple languages (Hatori et al. 2012; Zhang and Yang 2018). Taking this into consideration, our proposed framework is designed to perform joint training of such related tasks.

We propose a search-based structured prediction framework for numerous NLP tasks in a free word order language like Sanskrit. The framework we propose is an arc-factored model, similar to graph-based parsing frameworks (McDonald et al. 2005b; Ishikawa 2011). Here, the system expects a graph as input with its edges featurized, irrespective of the task. We design suitable inference procedures to incorporate

1 The International Alphabet of Sanskrit Transliteration (IAST) scheme, a lossless romanization scheme for Sanskrit. The International Phonetic Alphabet (IPA) equivalents for the IAST scheme can be found in `https://en.wikipedia.org/wiki/Help:IPA/Sanskrit`.

2 Nominative case can be *karma* in a passive construction, e.g., *dīpaiḥ nagarāṇi prakāśayante* – English translation: "Towns are illuminated by lamps"; here the *karma* '*nagarāṇi*' is in nominative case. Gloss: dīpaiḥ - Lamps; nagarāṇi - Towns; prakāśayante - Illuminate.

task-specific constraints, by which the search space for the possible solutions is considerably reduced. The task is then framed as the search for a task-specific structure. In principle, the graph-based dependency parsing approaches such as McDonald et al. (2005b) or the lattice-based morphological parsing approaches such as that of Kudo, Yamamoto, and Matsumoto (2004) can all be formalized as specific instances under this framework. To further elaborate, consider the case of dependency parsing. Here, the input graph will be a complete graph, with the (segmented) words in the sentence forming the nodes of the graph. Here, the specific sub-structure to search for will be a spanning tree (Hirakawa). The inference procedure searches for the minimum cost spanning tree, using a suitable algorithm such as Chu-Liu-Edmond (Edmonds 1967). Summarily, training consists of learning an energy function that assigns lower scores to the ground-truth spanning tree than the other candidate spanning trees. All our models follow an arc-factored approach, where the energy of the structure is nothing but the sum of the energies of its edges (Ishikawa 2011; LeCun et al. 2006). The edges being featurized, the energy function is used to score these featurized edge vectors.

The performance of a system depends highly on the choice of feature function used for the task. In MRLs, hand-crafted features still form a crucial component in contributing to the performance of the state-of-the-art systems for tasks such as morphological parsing and dependency parsing (More and Tsarfaty 2016; More et al., 2019; Seeker and Çetinoğlu 2015). But Krishna et al. (2018) learn a feature function using the Path Ranking Algorithm (PRA) (Lao and Cohen 2010) for the joint task of word segmentation and morphological parsing. PRA essentially maps the problem of learning a feature function to that of automatic learning of horn clauses (Gardner, Talukdar, and Mitchell 2015), where each clause is a morphological constraint. The domain knowledge required here confines to just defining the literals, the combinations of which will be used to form the clauses. In Krishna et al. (2018), morphological tags and grammatical categories form the literals and the feature (clause) values are calculated using distributional information from a morphologically tagged corpus. We find that the same feature function can be used effectively for all the standalone and joint tasks we experimented with, including the downstream morphosyntactic tasks. In the case of prosodification, prosody-level information, instead of the morphological information, is used to define the literals. We further improve our feature function learning approach using Forward Stagewise Path Generation (FSPG) (Meng et al. 2015). FSPG-based features consistently and significantly outperform PRA-based features and achieve state-of-the-art results in all the tasks we experiment with. Our work is an extension of the work by Krishna et al. (2018), where a joint model for word segmentation and morphological parsing was proposed. The contributions of our work are as follows:

1. We extend the work of Krishna et al. (2018) to a general graph-based parsing framework for multiple structured prediction tasks in Sanskrit. We achieve state of the art (SoTA) results in all the tasks we experiment with. In fact, this is the first work that introduces statistical models for performing dependency parsing and prosodification in Sanskrit.

2. We automate the process of learning a common feature function to be used across all the morphosyntactic tasks by using the FSPG approach. This is completely automated, and avoids the need for feature engineering for each task separately. Further, this simplifies the process of choosing the feature function, and is not constrained by the accessibility to domain expertise. We use the same approach for learning different feature functions for the prosodification task and the morphosyntactic tasks.

3. Sanskrit being a low-resource language, task-specific labeled data are particularly hard to come by. All our models use as low as 10% of the training data as required by the current neural SoTA models in Sanskrit for various tasks. We used around 9.26% (10,000) and 1.5% (8,200) of training data, as against 108,000 and 0.5 million training sentences for the SoTA neural models in syntactic linearization (Krishna et al. 2019) and word segmentation (Hellwig and Nehrdich 2018), respectively.

4. Our experiments on Sanskrit and Czech show the language-agnostic nature of the framework. We train models for three morphosyntactic tasks in Czech. We outperform all participating systems in the CoNLL 2018 shared task on "multilingual parsing from raw text to universal dependencies" (Zeman et al. 2018), that is, in joint morphological and dependency parsing. We report the third best score in comparison with the performance of participating systems in the SIGMORPHON 2019 shared task on "morphological analysis and lemmatization in context" (McCarthy et al. 2019), that is, in morphological parsing. Finally, we outperform two highly competitive neural dependency parsers (Qi et al. 2020; Straka and Straková 2017) in dependency parsing.[3]

5. Though the framework is language-agnostic, it still enables us to incorporate language-specific constraints to prune the input search space and filter the candidates during inference. Use of such constraints led to performance improvements in dependency parsing and syntactic linearization tasks in Sanskrit. The labeled attachment score [LAS] (unlabeled attachment score [UAS]) for dependency parsing improved from 79.28 (82.65) to 83.93 (85.32) and the BLEU score for syntactic linearization improved by about 8 BLEU scores.

This article is organized as follows. In Section 2, we first elaborate the characteristics and the history of usage of the language. Then we describe each of the tasks we perform along with the challenges that need to be addressed. Section 3 details the architecture of the EBM framework and then describes each component in the framework. Our experiments and results are discussed in Section 4. Section 5 discusses some of the key observations along with the future work. In Section 6, we present the related work. Section 7 then concludes the article by summarizing our key findings from the work. Table 1 provides the list of the most commonly used abbreviations in this work.

2. Computational Processing of Texts in Sanskrit and Its Challenges

Sanskrit is a classical language (Coulson 1976) and was the prevalent medium of knowledge transfer in the demographic of the Indian subcontinent for about three millennia (Pollock 2003; Goyal et al. 2012). Composition and preservation of Sanskrit texts as part of a rich oral tradition was a salient feature of the Vedic age, prevalent presumably in the second millennium BCE (Staal 2008; Scharf 2013). Fairly advanced disciplines of prosody (*Chandas*), phonetics (*śikṣā*), and grammar (*vyākaraṇa*), with their intellectual roots in the Vedic oral tradition, were developed for Sanskrit by the middle of the first millennium BCE (Scharf 2013; Kiparsky 1995). The oral tradition and these

3 All the experiments were performed on the Czech-PDT UD treebank.

Table 1
List of most commonly used abbreviations in this work.

Abbreviations			
SHR	Sanskrit Heritage Reader	WS	Word Segmentation
DCS	Digital Corpus of Sanskrit	MP	Morphological Parsing
PCRW	Path Constrained Random Walk	DP	Dependency Parsing
FSPG	Forward Stagewise Path Generation	SL	Syntactic Linearization
PRA	Path Ranking Algorithm	Raw2UD	CoNLL 2018 shared task on Raw Text to Universal Dependencies
EBM	Energy-Based Model	Joint T1 + T2	Joint modeling of the tasks T1 and T2
BoW	Bag of Words	MIT	Metre Identification Tool
MRL	Morphologically Rich Language	SoTA	State of the Art

later developments have shaped the characteristics of the language and its usage in multiple ways. First, the words in a sentence often undergo phonetic transformations at the juncture of their boundaries, similar to what one expects in connected speech. These transformations obscure the word boundaries and often result in the modification and the fusion of the sounds at the word boundaries (Matthews 2007, p. 353). Such transformations, called *sandhi*, are reflected in writing as well. Secondly, a large body of works in Sanskrit is in the form of verses, where a verse should adhere to one of the prescribed metrical patterns in Sanskrit prosody. Such constructions often followed a relatively free word order, with the grammatical information encoded via the morphological markers. The adherence to metrical pattern came even at the cost of verbal cognition to the listener (Bhatta 1990). However, later, commentators of these verses started to reorder the sentences in the "most easily understandable prose order" (Tubb and Boose 2007), or the "natural order" (Apte 1965), for easier verbal cognition. The prose so obtained is called the *anvaya* of the verses, and it is merely a permutation of the words in the verse. Both the original verse and its corresponding *anvaya* will have the same syntactic analysis as per the Sanskrit grammatical tradition. The word ordering in prose is observed to be more restrictive as it tends to follow SOV word order typology (Hock 2015). Further, the words tend to behave as constituents of phrases, implying that the phrases are continuous (Gillon and Shaer 2005; Schaufele 1991). Given this context and history of usage of Sanskrit, we describe the tasks we have considered.

2.1 Tasks

The tasks we consider are word segmentation, morphological parsing, dependency parsing, syntactic linearization (word ordering), and prosodification. In this section, we discuss each of these five tasks in detail. As shown in Figure 1, the aforementioned tasks are categorized into morphosyntactic tasks and prosody-level tasks. Prosodification, a prosody-level task, is the task of arranging a bag of words into a verse sequence, such that the resulting sequence follows a metrical pattern. Here, we do not make use of any morphosyntactic information about the input and rather use the syllable-level information for modeling the task. The rest of the tasks form a hierarchy such that each

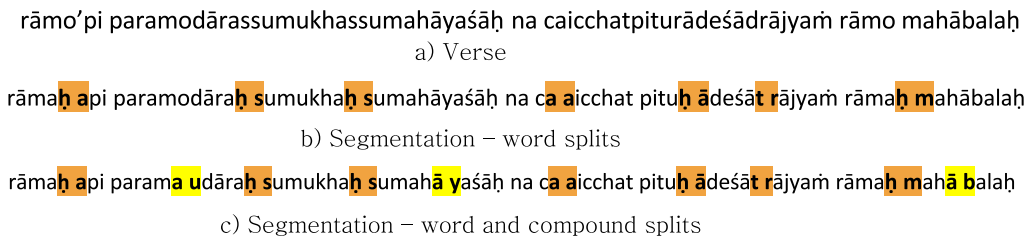

Figure 2
The reference sentence in (a) its original fused form, (b) its segmented form that shows all the word boundaries, and (c) its segmented form that shows all the word boundaries and compound splits. The regions where the *sandhi* originally occurred between word boundaries are shown in orange and where the *sandhi* originally occurred between the compound components are shown in yellow.

task expects predictions from its upstream tasks to fulfil its input requirements. The tasks in the hierarchy are collectively termed as morphosyntactic tasks as each of these tasks either require morphosyntactic information in input or is an upstream task to such tasks (e.g., word segmentation). The joint modeling of such related tasks in a hierarchy has empirically been shown to perform better than a pipeline-based approach (Seeker and Çetinoğlu 2015; More et al. 2019). We jointly model the morphological parsing (MP), and word segmentation (WS) tasks, similar to Krishna et al. (2018). We further extend this framework to jointly model WS, MP, and dependency parsing tasks as well as the WS, MP, and syntactic linearization tasks. Throughout the section we will use a verse from the literary work *Rāmāyaṇa*, "*Rāmo'pi paramodārassumukhassumahāyaśāḥ na caicchatpiturādeśādrājyaṃ rāmo mahābalaḥ*," for illustrative purposes. We will henceforth refer to this sentence as the "reference sentence." The sentence translates to, "But Rama, a source of universal delight, exceedingly noble, of beautiful countenance, of very great fame, very strong, did not want to accept the kingdom in accordance with the command of his father."[4]

Word Segmentation. Word segmentation is the task of identifying the words in a given character sequence. This can be a challenging task in a language like Sanskrit, where the word boundaries are often obscured due to *sandhi*. *Sandhi* is defined as the euphonic assimilation of sounds (i.e., modification and fusion of sounds) at or across the boundaries of grammatical units (Matthews 2007). Although such phonetic transformations between the morphemes of a word are common across languages, these transformations are observed also between the successive words in a sentence in Sanskrit. For instance, Figure 2(b) shows the verse with all the seven splits of the words that were fused due to *sandhi* in its original form (Figure 2(a)).[5] Here, six of the seven instances of *sandhi* result in phonetic transformations at the word boundaries, while the remaining one results in concatenation of the words without any phonetic transformations. Similarly, Figure 2(c) shows the splits between the components of a compound along with the word splits. Here, we jointly perform the task of compound splitting, along with word

4 Gloss: rāmaḥ - one who delights everybody, api - also, paramodāraḥ - exceedingly noble, sumukhaḥ - beautiful countenance, sumahāyaśāḥ - greatly renowned, mahābalaḥ - very strong, Rāmaḥ - Rama (Name of a person), pituḥ - father's, ādeśāt - by command, rājyaṃ - kingdom, na - not, ca - and, aicchat - desired.

5 For example, c**a** + **a**icchat → c**a**icchat; pitu**ḥ** + **ā**deśāt → pitu**r**ādeśāt; ādeśā**t** + **r**ājyaṃ → ādeśā**d**rājyaṃ. For a complete list of possible transformations due to *sandhi*, visit `https://bit.ly/2BFy01F`.

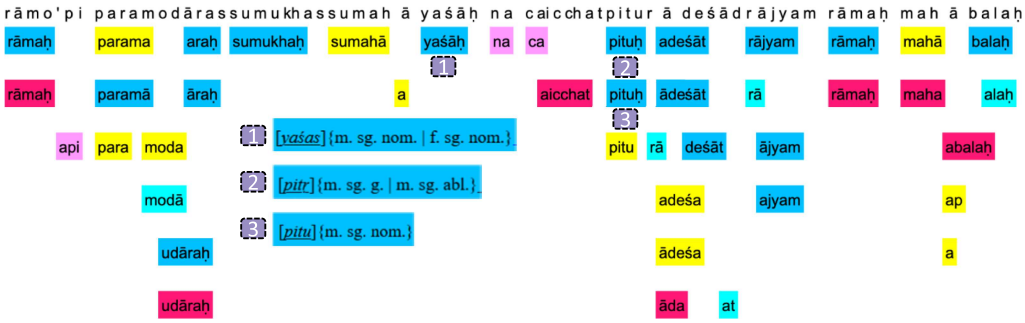

Figure 3
All the lexically valid segmentations for the reference sentence based on the analysis from SHR. The candidate segments are color coded by SHR based on their lexical categories. Blue for substantives, red for finite verb-forms, mauve for indeclinables, and yellow for all the non-final components of a compound. Cyan is used for those inflected-forms that can only be used as the final component of a compound (Goyal and Huet 2016). The numbered boxes indicate the morphological analysis as per SHR for the inflected forms *yaśāḥ and pituḥ*.

segmentation, similar to previous word segmentation models in Sanskrit (Hellwig and Nehrdich 2018; Reddy et al. 2018). The knowledge of the individual components of a compound will help in its analysis in downstream tasks, and hence is important for processing Sanskrit corpora abundant with multicomponent compounds.

The analysis of a sequence with fused words can lead to ambiguity in identifying the original words in the sequence. Goyal and Huet (2016) propose Sanskrit Heritage Reader (SHR), a lexicon driven shallow parser that encodes all the rules of *sandhi* as per traditional Sanskrit grammar.[6] SHR can enumerate all possible lexically valid segmentations for a given sequence. Figure 3 shows the possible analyzes for the reference sentence as per SHR.[7] Here, we define a segmented solution that spans the entire input sequence as an "exhaustive segmentation." For the sentence under consideration, the correct solution is one among the 59,616 possible exhaustive segmentations (Section 2.1). Given the possible word splits, our task can be formalized as one that finds the semantically most valid exhaustive segmentation among the candidate solutions.

Morphological Parsing. Morphological parsing is the task of identifying the morphemes of the words in a sentence. Specifically, our task focuses on obtaining the correct stem and the morphological tag of the inflected forms in a sentence. Sanskrit, similar to Czech (Smith, Smith, and Tromble 2005), is a fusional language where a morpheme encodes multiple grammatical categories. Morphological parsing in Sanskrit is challenging primarily because of two factors. First, Sanskrit has a rich tagset of about 1,635 possible tags. Table 2 shows the lexical categories in Sanskrit and the grammatical categories they comprise of. Second, an inflected form in Sanskrit may lead to multiple morphological analyzes, due to syncretism and homonymy. For instance, Table 3 shows the candidate morphological analyzes produced by SHR for the inflected-forms *pituḥ* and *yaśāḥ*.[8]

6 `https://sanskrit.inria.fr/DICO/reader.fr.html`.
7 The analysis is available at `https://bit.ly/2WYVkie`.
8 Figure 3 also shows the SHR analysis (marked with numbered boxes) for both the inflected forms; *Yaśāḥ* is the final component of the compound *sumahāyaśāḥ*. In Sanskrit, the inflectional marker is applied generally to the final component of a compound.

Table 2
Grammatical features and their distribution over inflectional classes. "Other" includes forms such as infinitives, absolutives, compound-components, indeclinables, etc.

Feature	Values	Noun	Fin. verb	Participle
Tense	18		✓	✓
Case	8	✓		✓
Number	3	✓	✓	✓
Gender	3	✓		✓
Person	3		✓	
Other	6			
Total	41	72	162	1,296

Table 3
Instances of homonymy and syncretism in the reference sentence. *yaśāḥ* is a case of syncretism, where the stem has the same inflected form but has different morphological tags. *pituḥ* is a case of both syncretism and homonymy, as it can be an inflection of two different stems, *pitu* and textitpitṛ. "Num" is the grammatical category, "number." The information is based on the analysis from SHR.

Word	Stem	Morphological Tag		
		Case	Num	Gender
yaśāḥ	yaśas	nominative	1	feminine
		nominative	1	masculine
pituḥ	pitṛ	genitive	1	masculine
		ablative	1	masculine
	pitu	nominative	1	masculine

Here, the word *pituḥ* has 3 possible analyzes, of which two are cases of syncretism for the inflected-forms of the stem *pitṛ* and the third analysis is an inflection of the stem *pitu*. Similarly, *yaśāḥ* is a case of syncretism, where it has two possible analyzes both with the same stem *yaśas*, but differing in their morphological tags. In this task, we rely on SHR to obtain an exhaustive list of possible morphological analyzes for the words in a sequence. We then formulate our task as obtaining the correct morphological analysis from the exhaustive list of candidate solutions obtained from SHR.

Dependency Parsing. Given a sentence in Sanskrit, dependency parsing requires finding the syntactic relations between the words in the sentence, thereby predicting a labeled dependency tree as the final output. For the task, we follow the widely adopted dependency tagging scheme proposed for Sanskrit (Kulkarni, Pokar, and Shukl 2010; Kulkarni and Ramakrishnamacharyulu 2013). The tagging scheme, consisting of 22 relations,[9] is in principle motivated from the traditional dependency analysis for Sanskrit, known as the *kāraka* theory (Kiparsky and Staal 1969). These relations are known to be syntactic-semantic in nature (Bharati and Sangal 1993). Using this scheme enables us to integrate our predictions into the pipeline of systems currently in use for linguistic annotations and processing of Sanskrit texts (Goyal et al. 2012; Huet and Kulkarni 2014; Goyal and Huet 2016; Das 2017). The relations rely heavily on the case markers of the nominals and the valency of the verb to infer the structural information of the sentence (Kiparsky and Staal 1969; Ramkrishnamacharyulu 2009). Figure 4 shows the dependency analysis for the reference sentence. For presentational clarity, the figure uses the prose word ordering (*anvaya* of the verse) rather than the original verse order. The sentences in prose

9 https://bit.ly/3hKLZT9.

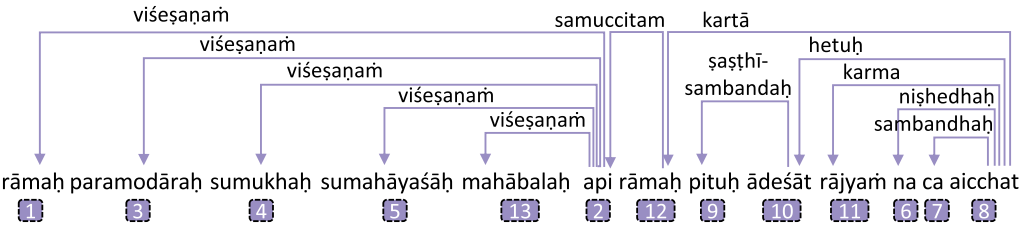

Figure 4
Dependency analysis for the reference sentence. The *kāraka* tags as per the dependency tagging scheme of Kulkarni, Pokar, and Shukl (2010) are shown as the edge labels in the figure. For presentational clarity, the figure uses the word ordering from the *anvaya* of the verse. The numbers in the boxes indicate the position of the word (from left) in the original word order in the verse. The corresponding English translation for the tags are: Hetuḥ – Cause; Karma – Object; Kartā – Subject; Niṣedhaḥ – Negation; Sambandhaḥ – Relation; Samuccitam – Conjunction; Saṣṭhīsamsambandhaḥ – Genitive or possessive relation; Viśeṣaṇam – Adjectival modifier.[10]

in Sanskrit tend to follow weak non-projectivity in their dependency analyzes, but the same is not guaranteed for the word arrangements in verse order (Kulkarni et al. 2015). Nevertheless, the dependency tree is not dependent on the configurational information of the words in a sequence.

Syntactic Linearization. Commentators often reorder the words in a verse and place them in the "most easily understandable prose order" (Tubb and Boose 2007, p. 150), as part of writing a commentary for the verse. This converted prose order is called the *anvaya* of the verse, and Apte (1965) describes it as the natural order or connection of words in a sentence, construing grammatical order or relation. Such orderings tend to follow a subject-object-verb word order (Hock 2015; Tubb and Boose 2007) and facilitate easier verbal cognition of the sentence for others (Bhatta 1990). As previously mentioned, the words in an *anvaya* tend to behave as constituents of phrases, implying that the phrases are continuous (Gillon and Shaer 2005; Schaufele 1991). The objective of this task is to predict the *anvaya* of a given verse. However, the word order in a verse is guided by the metrical constraints, and has little to offer in rearranging the words to a prose order[11] (Scharf et al. 2015; Kulkarni et al. 2015). Hence we formulate the task as a syntactic linearization task. Syntactic linearization is the task of ordering a bag of words (BoW) into a grammatical and fluent sentence (Liu et al. 2015). In the standalone setting, we consider a BoW, along with correct morphological analysis for each of the words in the BoW, as the input. Figure 5 shows an instance of syntactic linearization from a BoW. But a more realistic scenario is when the input is a verse in its original written form, where several words may be present in the fused form due to *sandhi*. In the joint setting, the task takes the verse in its original written form and jointly performs word segmentation, morphological parsing, and linearization. The word order divergences between the verse and the corresponding *anvaya* at a syntactic level can be assessed using the following three aspects. First, Sanskrit sentences in their prose order tend to follow SOV typology (Hock 2015). In the reference sentence, as shown in Figure 4,

10 "Sambandhaḥ" translates to relation. This coarse-level tag is included in the tagging scheme, to assign to those cases which require extra-syntactic factors for resolving the exact fine-grained relation. For a more detailed understanding of the tagging scheme and *kāraka* theory in general, please refer to Kulkarni and Sharma (2019) or Kulkarni, Pokar, and Shukl (2010).

11 We use the terms *anvaya* and prose order interchangeably in this work.

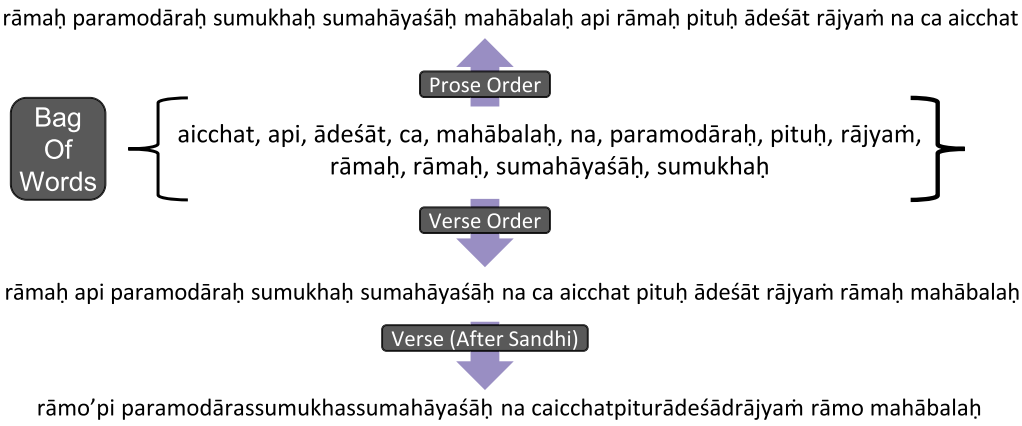

Figure 5
Instances of syntactic linearization and prosodification from a BoW input. In prosodification, *sandhi* may be essential at specific word boundaries to adhere to metrical constraints. However, *sandhi* is purely optional in sentences in prose order.

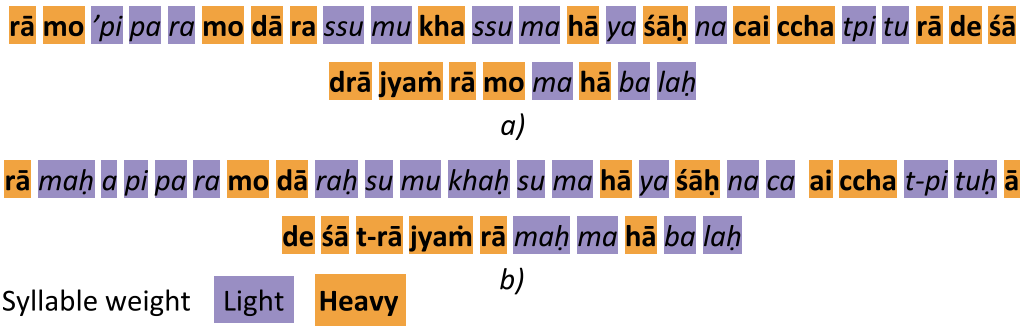

Figure 6
Syllable-level segmentation, along with the syllable weights for (a) verse in its original form; (b) verse with segmented word boundaries.

rāmaḥ (numbered 12) is the *kartā* (subject), *rājyaṃ* (numbered 11) is the *karma* (object), and both have the verb *aicchat* (numbered 8) as their head. As shown in Figure 5, the prose order conforms to the SOV typology, whereas the verse order does not. Second, dependency analysis of the prose constructions in Sanskrit tends to be planar, or weakly non-projective (Kulkarni et al. 2015). Finally, the ordering in the prose is likely to follow the principles of dependency locality theory (Gibson 1998, DLT). However, none of these three claims can be established for a verse. As per DLT, the linear distance between words linked in dependencies should be as short as possible (Gibson et al. 2019). As a consequence of the DLT, the dependency length of the prose ordering tends to be shorter than that of the verse ordering.

Prosodification. We define this task as imposition of prosodic structure to a BoW input. We specifically use the metre information, based on Sanskrit prosody, to convert a BoW input into a verse sequence. For a verse sequence, its metre can be deterministically identified based on the sequence of syllable weights present in the sequence. The syllable weight can either be *laghu* (light) or *guru* (heavy), decided deterministically using

rule-based systems (Melnad, Goyal, and Scharf 2015). Our task is to convert a BoW input to a sequence such that the generated sequence adheres to one of the prescribed metres in Sanskrit prosody. While prosodification involves rearranging a BoW to a sequence, the rearrangement in itself need not result in a sequence that adheres to a valid metrical pattern. Given a sequence, the phonetic transformations due to *sandhi* may lead to reduction in the number of syllables in the sequence or might alter the syllable weight of one or more syllables. Although *sandhi* operations are optional when writing in prose, a poet might be compelled to use *sandhi* in a verse to obtain a valid sequence of syllable weights (i.e., a metrical pattern). Figure 5 shows the linear arrangement of the words followed in the verse, as well as the final verse sequence where *sandhi* is performed at specific word boundaries. While the former does not adhere to any known metre in Sanskrit prosody, the latter belongs to the *Anuṣṭubh meter*. So in prosodification, our task is not just confined to finding the correct permutation from a BoW, but also to determine the junctures at which *sandhi* needs to be performed. As shown in Figure 6, the reference sentence has 32 syllables in its original form (Figure 6(a)). Further, its syllable weight pattern adheres to the Anuṣṭubh meter. However, the segmented form, as shown in Figure 6(b)), results in a sequence of 34 syllables. The segmented sequence, with 34 syllables, does not adhere to any known meter pattern in Sanskrit prosody. Consider the string, *rāmo'pi*, a substring of the reference sentence. *rāmo'pi* has 3 syllables, where the syllables *rā* and *mo* are heavy syllables and the remaining *'pi* is a light syllable. However, after word segmentation, the string becomes *rāmaḥ api*; this has 4 syllables, where *rā* is a heavy syllable, while *maḥ*, *a*, and *pi* are light syllables. This illustrates how *sandhi* becomes a convenient tool for poets in verse generation and the decision of *sandhi* is bound by the metre patterns. In the case of prose, since the generated sentence is not bound to follow any metrical pattern, *sandhi* is purely optional.

3. Energy-Based Framework for Structured Prediction in Sanskrit

We define a search-based structured prediction framework using EBMs (LeCun et al. 2006) for performing numerous sequence-level NLP tasks in a free word order language like Sanskrit. The framework essentially consists of three components, namely, a graph generator, an edge vector generator, and a structured prediction model. A structured prediction model can further be subdivided into input representation, learning, and inference (Belanger 2017). The processed outputs from the graph generator and edge vector generator form the input representation for the structured prediction model. For all the tasks we consider, the graph generator takes input from the user, which can either be a sequence or a BoW, and converts it into a graph structure. The edge vector generator then generates feature vectors for the edges in the graph.

Formally, the aforementioned graph and feature vectors form the observed variable X to the structured prediction model in our framework. The model performs inference on X to predict an induced subgraph of X, which forms the output variable Y. Here, X and Y are structured objects that can be factorized into multiple variables. Such problems are challenging because the number of candidates in the output space is exponential in the number of output variables that constitute the factored structure of Y (Doppa, Fern, and Tadepalli 2014; Belanger 2017). EBMs enable non-probabilistic training of structured models, thereby avoiding the need to normalize over Y. EBMs can be used to design architectures that can incorporate known properties about the language or the properties beneficial for the task, and then perform constrained optimization over Y (Belanger, Yang, and McCallum 2017). In EBMs, a model is viewed as an energy function that captures the dependencies between the observed and output

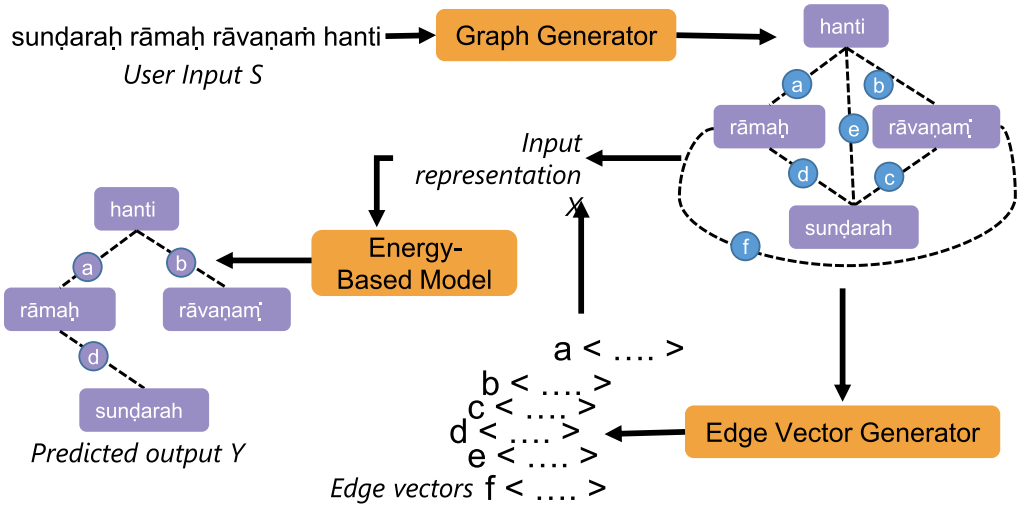

Figure 7
Overview of the EBM architecture. Dependency parsing for a four word sentence, *sundaraḥ rāmaḥ rāvaṇam hanti* [Handsome Rama kills Ravana], is shown as the use-case.

variables, by associating a scalar energy to each configuration of the variables (LeCun et al. 2006). The inference process consists of finding the values of output variable Y that are most compatible with the observed variable X, such that the energy is minimized (LeCun et al. 2007). Learning consists of finding an energy function that associates lower energies to correct output structures and higher energies to the incorrect ones (LeCun et al. 2007). The models we train are non-probabilistic models that use "margin losses" (LeCun et al. 2006, §6) to create the energy gap between the correct structure and incorrect structures.

An overview of our framework is shown in Figure 7. In the figure, dependency parsing for a four word sentence S, *sunḍaraḥ rāmaḥ rāvaṇaṁ hanti* [Handsome Rama kills Ravana], is shown for illustrative purposes. Similar to other graph-based parsing approaches, S is first converted to a complete graph using graph generator. The edge vectors are then generated such that each edge captures the distributional information between the pair of nodes it connects. The graph structure and the edge vectors together form the input (X) to the EBM, which uses an inference procedure to predict a structure with the minimum energy (Y). For dependency parsing, Chu–Liu–Edmond's algorithm becomes the inference procedure that searches for the minimum energy spanning tree.[12]

We train eight models, five in standalone setting, and three in joint setting, for the tasks mentioned in Section 2.1. While the learning procedure generally remains the same across the tasks in the framework, the input representation and the inference procedures are task-specific. Table 4 enumerates the setup for each of the tasks in terms of user given input S, the observed variable X, the output variable Y, and the inference applied. In our case, X will always be a graph $X(V_X, E_X)$.[13] Here, V_X forms the vertex

12 We search for the minimum energy spanning tree, instead of the maximum spanning tree as in the case of McDonald et al. (2005b).

13 We overload the notations to indicate both the observed variable for the model as well as its corresponding graph representation.

Table 4
Task-wise description of user input S, observed variable of the model X, output variable of the model Y, and the inference applied.

Task	User Input S	Observed Variable X	Output Variable Y	Vertex Attributes $\mathcal{A}$	Inference
Morphosyntactic tasks					
Word Segmentation (WS)	A string with fused words	Graph V_X = set of words	Maximal Clique	A_1 = Stem A_2 = Inflected form A_3 = Morph. Tag	Greedy maximal clique selection heuristic. §3.3, Algorithm 1
Morphological Parsing (MP)	A sequence of segmented words	Graph V_X = set of words	Maximal Clique		
Dependency Parsing (DP)	A sequence of segmented words, with morphemes	Complete Graph V_X = set of words	Spanning Tree		Chu–Liu–Edmonds Chu and Liu (1965) Edmonds (1967)
Syntactic Linearization (SL)	A bag of words, with morphemes	Complete Graph V_X = set of words	Hamiltonian Path		Beam Search
Joint WS + MP	A string with fused words	Graph V_X = set of words	Maximal Clique		§3.3, Algorithm 1
Joint WS + MP + DP	A string with fused words	Graph V_X = set of words	Steiner Tree		Prim's algorithm §3.3 Algorithm 2
Poetry to Prose Joint WS + MP + SL	A verse (adhering to metre) with fused words	Graph V_X = set of words	Hamiltonian Path		Beam Search
Prosody level task					
Prosodic Linearization (PL)	A bag of words	Complete Graph V_X = set of syllables	Hamiltonian Path	A_1 = Syllable A_2 = Syllable Weight	Beam Search

set and E_X forms the edge set of the graph X. Each vertex encodes relevant linguistic information about the component it represents. For morphosyntactic tasks, V_X is a set of words, and syllables form the vertices for the prosody-level tasks. Every edge $e \in E_X$ is then featurized using the "edge vector generator" module. A suitable inference procedure gives a structured output Y^* with the minimum energy, from a set of possible solutions $\mathcal{Y}$ (LeCun et al. 2006). Y^* can be expressed as:

$$Y^* = \underset{Y \in \mathcal{Y}}{\operatorname{argmin}}\, \mathcal{E}(Y, X)$$

where $\mathcal{E}$ is the energy function and $\mathcal{Y}$ is the set of possible solutions. Every element $Y \in \mathcal{Y}$ is an induced subgraph of X. The design of the inference procedure further restricts the type of induced subgraphs that can form a solution, and this is task-specific. For instance, in word segmentation, we use the greedy maximal clique selection approach as the inference (Section 3.3, Algorithm 1), thereby restricting $\mathcal{Y}$ to be the set of all maximal cliques in X. Similarly for dependency parsing, $\mathcal{Y}$ is the set of all the directed spanning trees in X. Ours is an arc-factored model and, hence, the energy of the structure produced by the inference procedure is factorized as the summation of energies of its edges (McDonald et al. 2005b; Ishikawa 2011).

$$\mathcal{E}(Y) = \sum_{e \in E_Y} \mathcal{E}(\vec{e})$$

Here $\vec{e}$ is a non-negative real valued vector, for each edge $e \in E_X$. The energy function $\mathcal{E}(.) : [0, \infty)^{|\vec{e}|} \rightarrow (-\infty, \infty)$, takes non-negative real valued edge-vector and produces a scalar energy value. The energy function is calculated using multilayer perceptrons, with leaky ReLU activation function at the hidden layer. The training involves learning an energy function that minimizes the energy of the ground-truth structure Y^{GT} as compared to other candidate structures in $\mathcal{Y}$. We apply hinge loss (Taskar, Guestrin, and Koller 2003; Altun, Johnson, and Hofmann 2003), a generalized margin loss, as the loss function. The loss function takes the following general form:

$$L = max(0, m + \mathcal{E}(Y^{GT}) - \mathcal{E}(\bar{Y}))$$

where m is the positive margin, $\mathcal{E}(Y^{GT})$ is the energy of the ground truth solution, and $\mathcal{E}(\bar{Y})$ is the energy of the incorrect candidate with the lowest energy. This loss attempts to make $\mathcal{E}(Y^{GT})$ lower than $\mathcal{E}(\bar{Y})$, at least by m (LeCun et al. 2007, 2.1). Thus, unless $\mathcal{E}(\bar{Y})$ has the energy value larger than $\mathcal{E}(Y^{GT})$ by the margin m, loss is computed. The margin is defined as a function that captures the structural divergences between the ground truth and other candidate structures, such that energy of an incorrect candidate with a higher structural divergence should be much higher. For dependency parsing, the margin m is the number of nodes with an incorrect head attached to them (McDonald et al. 2005b; Carreras 2007; Bohnet 2010). Similarly, for all the tasks that use beam-search as inference and for joint WS+MP+DP, the margin is the number of incorrect local decisions, that is, incorrect edges. For word segmentation, morphological parsing and also for joint WS+MP (Krishna et al. 2018), we predict a structure containing a subset of nodes from the vertex set V_X of the input X. Here, the margin is the square of the number of nodes in the prediction that are not in the ground truth, namely,

$$m = |V_{\bar{Y}_T} - V_{Y^{GT}}|^2$$

We minimize the given loss function using gradient descent. The network parameters are updated per sentence using back-propagation. The hinge loss function is not differentiable at the origin. Hence, we use the subgradient method to update the network parameters (Socher, Manning, and Ng 2010; Ratliff, Bagnell, and Zinkevich 2007). Next, we discuss various components of our framework in further detail.

3.1 Graph Generator

The graph generator analyzes the user input S, and transforms S into a graph $X(V_X, E_X)$. An edge is added between every two vertices that can co-exist in a predicted output structure. Hence the set E_X encodes the structural information between the vertices in V_X. The edges are featurized to capture the distributional information between the pair of vertices they connect. This feature vector is used to calculate the scalar energy for the edge, which indicates the strength of the association between the node pairs. For morphosyntactic tasks, we have word-level vertices in V_X. Here, each vertex is a word, represented using the word's inflected form, its stem, and its morphological tag. Similarly, prosody-level tasks have syllable-level vertices, where each vertex encodes the syllable and its syllable weight. Thus, a vertex v in the set V_X essentially represents some

linguistic information encoded as a tuple of multiple attribute–value pairs.[14] Formally put, every (type of) task consists of a predetermined set of attributes, $A_1, A_2 \ldots A_n$. We define $\mathcal{A}$ as the Cartesian product of these sets, $\mathcal{A} = A_1 \times A_2 \ldots \times A_n$. Every vertex encodes a tuple $a \in \mathcal{A}$, where $a(i)$ indicates the value for the i^{th} attribute. The vertices in V_X for morphosyntactic tasks form a 3-tuple, consisting of (*inflected form*, *stem*, *morphological tag*), while that of a prosody-level task form a 2-tuple, consisting of (*syllable*, *syllable weight*).

Morphosyntactic Tasks. As shown in Table 4, we experiment with seven different settings for the morphosyntactic tasks, of which four are for standalone and three are for the joint models. For standalone dependency parsing and syntactic linearization, the user input S is expected to include the gold-standard inflected form, stem, and morphological tag for each word in the sentence. This information is used directly to construct the vertices and the aforementioned attribute tuples (3-tuple) for these vertices. However, for other tasks, the user input S is expected to be a sequence. The sequence is then analyzed using a lexicon-driven shallow parser, Sanskrit Heritage Reader (Goyal and Huet 2016, SHR). SHR relies on finite state methods to exhaustively enumerate all possible word splits for this sequence, based on rules of *sandhi*. As previously shown in Figure 3 (§2.1), SHR not only generates possible candidate words, but it also provides the word-level morphological analysis for each candidate word it generated. SHR is limited only by the coverage of its lexicon's vocabulary. The SHR analysis is then used to form the vertex set V_X and 3-tuple of attributes for each vertex in V_X. For instance, SHR provides four separate analyzes corresponding to the substring *pitur* in the reference sentence, as shown in Figure 3. One analysis is for the compound component (in yellow box) *pitu*, and the remaining three analyzes accommodate the homonymy and syncretism expressed by the surface-form *pituḥ* as per Table 2.[15] Thus it will be represented by four different nodes in the input graph X. For morphological parsing, as the task does not involve word segmentation, the user input S will be a sequence of segmented words, for which SHR will produce only the stem and tag analyzes. Summarily, for all the seven morphosyntactic tasks, as shown in Table 4, each unique combination of the aforementioned three attributes forms a vertex in X.[16]

The edges in X should connect those nodes that can co-occur in a solution. If two candidate words are proposed as alternatives, such as the words *ādeśāt* and *deśāt* in Figure 3, then they are defined as *conflicting nodes*. In Figure 3, we can find that all the conflicting nodes do have overlapping spans in their word position with respect to the input sequence. An exception to this will be those overlaps valid under the *sandhi* rules, such as the pair of words *ca* and *aichhat* overlapping at the input character *a* in Figure 3. The edge set E_X consists of edges between every pair of nodes that are not conflicting. This implies that a complete graph will be formed for standalone dependency parsing and syntactic linearization, as there will be no conflicting nodes in the input for both tasks. For other tasks, our current design choice results in a denser graph structure as input. Such a design choice may lead to a computationally costly inference, which requires justification. A large number of texts in Sanskrit are written in verses. The

14 We additionally add unique identifiers to each of the vertices, in order to avoid confusion due to multiple instances of the same tuple in the input.

15 The morphological analyzes for *pituḥ* are shown also in Figure 3, marked with numbered boxes 2 and 3.

16 We additionally consider the span of the word-form with respect to the input sequence (as a unique identifier) to distinguish between words that are used more than once in the sentence. For instance, there are two occurrences of the inflected-form *rāmaḥ* in the reference sentence used in §2.1.

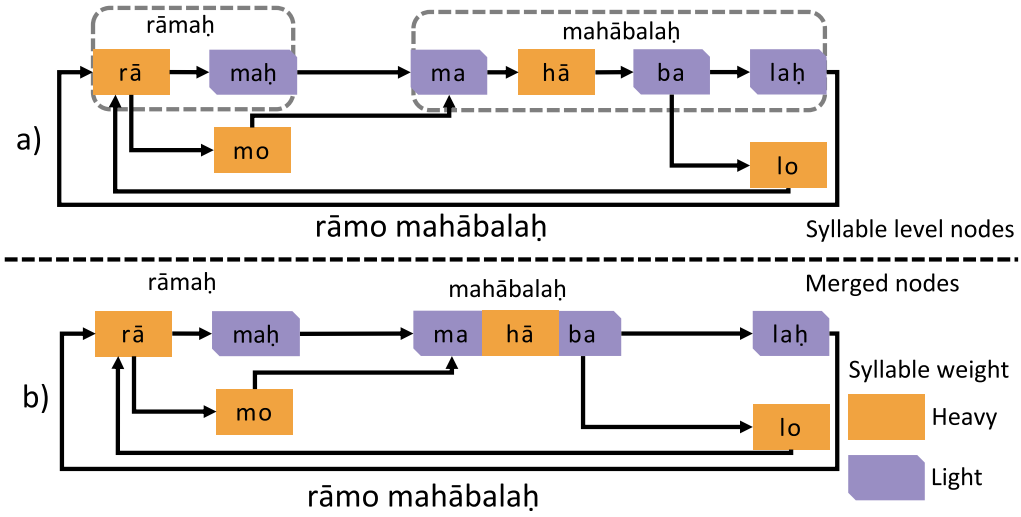

Figure 8
Input representation for the prosody-level task. Nodes where there is no ambiguity in traversal are merged to form a single node in the merged node representation.

relatively free word order structure of Sanskrit, especially in verses, motivated us to design a denser graph structure. Such a construction is agnostic to the original word order in a sentence and ignores any word order information present in the original input. The denser graph structure we use enables us to capture the relation between word pairs that might be grammatically related, even though the ordering between the words may not be indicative of this.

Prosody-Level Task. Prosodification is the sole prosody-level task that we discuss in this work. In this task, a BoW input is converted into a sequence that adheres to one of the prescribed metres in Sanskrit. In addition to finding a suitable word order for the words in the BoW, the task involves identifying the word boundaries where *sandhi* needs to be applied. The phonetic transformation due to *sandhi* may alter the syllable patterns of the sequence. For the task, the user given input S, namely, a BoW, is converted into a syllable-level graph $X(V_X, E_X)$. Here, each node encodes a syllable, representing the surface-form of the syllable and its syllable weight. The syllables and the corresponding weight of these syllables for each word are identified deterministically using the Metre Identification Tool by Melnad, Goyal, and Scharf (2015). Figure 8(a) shows the syllable level graph X for a two word BoW input, *mahābalaḥ* and *rāmaḥ*. The vertex set V_X for this input is initialized with all the syllables of both the words, as shown in the boxes enclosed with dotted lines in Figure 8(a). The graph is augmented further, by keeping the following three points in mind. First, the relative ordering of syllables in a word should not be allowed to change, as that would lead to a different word altogether. This is ensured by representing each word as a directed path, namely, *rā*→*maḥ* and *ma*→*hā*→ *ba*→ *laḥ*. For a word, the path begins with the first syllable of the word and terminates at the last syllable of the word. Second, it should be possible to predict any possible permutation of words in the input BoW. To ensure this, we form directed edges from the last syllable of every word to the first syllable of all the remaining words

in the input.[17] The current input can lead to two permutations, *rāmaḥ mahābalaḥ* and *mahābalaḥ rāmaḥ*. To enable these two permutations, we add edges from the last syllable nodes of each word, *maḥ* and *laḥ*, to the first syllable nodes of each word, *ma* and *rā*, respectively. Third, the final sequence that adheres to a metre might have to undergo *sandhi* at specific word boundaries. This leads to two more possible sequences, *rāmo mahābalaḥ* and *mahābalo rāmaḥ*, one each from either of the two permutations. Hence, two more syllable nodes are added, namely, *mo* and *lo*. *mo* is inserted between *rā* and *ma* and *lo* is inserted between *ba* and *rā*. The graph constructed after following all the three points can be seen in Figure 8(a). The graph allows us to generate any valid permutation of the input BoW, including all the possible sequences of each permutation that can be generated due to *sandhi*. We use beam search as our inference procedure. However, if we look into the directed path *ma*→*hā*→ *ba*, the inference has no prediction to make here. We will refer to such paths as unambiguous paths. The nodes in an unambiguous path can be merged together to form a single node, and this helps to reduce the search space size. The resultant graph after merging of unambiguous paths is shown in Figure 8(b).[18]

3.2 Edge Vector Generator

Each edge in the edge set E_X is passed onto the edge vector generator to obtain its feature vector. The edge vectors, along with the graph structure, form the input to the arc-factored structured prediction model. Identifying a set of features that is beneficial for each task is a challenge in itself. Morphologically rich languages, the state-of-the-art models for dependency parsing (More et al. 2019; Seeker and Çetinoğlu 2015), and even morphological parsing (More and Tsarfaty 2016), rely on hand-crafted features to obtain the feature function. In our case, we automate the learning of the feature function by using Forward Stagewise Path Generation (FSPG) algorithm (Meng et al. 2015). The edge vector generation in Krishna et al. (2018) for the joint word segmentation and morphological parsing task was performed using Path Ranking Algorithm (Lao and Cohen 2010, PRA). Both PRA and FSPG essentially map the problem of learning a feature function to that of automatic learning of arbitrary length horn clauses (Gardner, Talukdar, and Mitchell 2015). Both approaches follow a two-step process of feature function learning and feature value computation. Feature function learning involves automated generation and selection of a finite subset of features from a potentially infinite feature space, which are beneficial to the task. For all the morphosyntactic tasks, we learn a common feature function and a separate feature function is learned for the prosody-level task. This is a one-time process that happens prior to the training of the models. Feature-value computation involves finding the feature values for every edge, and hence it happens every time an input is processed.

For feature value computation, both FSPG and PRA use Path Constrained Random Walks (Lao and Cohen 2010, PCRW), a random walk–based inference approach. However, both approaches differ in the feature function learning step. PRA requires exhaustive enumeration of all the possible horn clauses in the feature space prior to

17 As a converse, this implies that the first syllable of each word receives an edge from the last syllable of all the other words in the input.

18 In Figure 8(b), though *laḥ*→*rā* forms an unambiguous path, we will not merge the nodes in the path. This is because the nodes in the path do not belong to the same word. Further, we generally have multiword inputs and in such cases there will be more than one outgoing edges from the syllable nodes at the word boundaries.

feature selection. An arbitrary limit on the maximum number of predicates possible in a horn clause is set, to make the feature space finite. The feature enumeration step is followed by feature selection and feature value computation steps. FSPG, on the other hand, is a modified version of least angle regression, which uses greedy strategies for feature selection, thereby avoiding the need for exhaustive enumeration of the features. For a given edge, a feature should essentially capture the strength of the association between the two nodes the edge connects, that is, how likely the two nodes are to co-occur in a solution. We calculate the strength of the association between the nodes based on distributional information obtained from a corpus. Each feature acts as a means to calculate this information under specific linguistic constraints. We now elaborate on our feature space, feature value computation approach using PCRW, and finally the feature function learning approach using FSPG.

The Feature Space. We define the feature space $\mathcal{F}$, from which the features, used for forming the edge vectors, are filtered. Given an edge in E_X, let the corresponding attribute tuples for the pair of nodes it connects be denoted as $a, a' \in \mathcal{A}$. For the morphosyntactic tasks, an element in $\mathcal{A}$ (i.e., an attribute tuple) is a 3-tuple consisting of (*inflected form*, *stem*, *morphological tag*), while for a prosody-level task it is a 2-tuple, consisting of (*syllable*, *syllable weight*). Now, $a(i), a'(j)$ are the i^{th} and j^{th} indexed attributes of a and a', respectively. A feature $f \in \mathcal{F}$ can be defined as a triple $\langle a(i), \psi, a'(j) \rangle$, which captures the distributional information between the attributes $a(i)$ and $a'(j)$, under the presence of the constraint ψ. Concretely, consider an edge connecting the node with the attribute tuple (*pituḥ*, *pitṛ*, *genitive|1|masculine*)[19] to a node with the attribute tuple (*rāmaḥ*, *rāma*, *nominative|1|masculine*). So, a feature $\langle a(1), \psi, a'(2) \rangle$ should capture the distributional information between the inflected form *pituḥ* and the stem *rāma*, namely, $\langle pituḥ, \psi, rāma \rangle$, under the presence of the constraint ψ.

Formally, a feature, $f = \langle a(i), \psi, a'(j) \rangle$, is a tuple $f = (a(i), \psi(1), \psi(2), \ldots\ldots \psi(k), a'(j))$ of size $k+2$, where k can be any non-negative integer. Every intermediate entry $\psi(1), \psi(2), \ldots\ldots \psi(k)$ is an entry in the constraint tuple ψ. For morphosyntactic tasks, we use morphological constraints and for prosody-level tasks, we use subpatterns of a metre (sequences of syllable weights) as these constraints. Because ψ can also be an empty tuple, it is possible to enumerate nine (four) different features for a morphosyntactic task (prosody-level task), with no constraints. This comes from the possible combinations of attributes from $a, a' \in \mathcal{A}$. We refer to this type of feature as ϵ-features, $\langle a(i), \epsilon, a'(j) \rangle$. But, the total number of possible features can be infinite, as the morphological constraint ψ can be an arbitrary length tuple. We now define ψ as a member of a countably infinite set $\mathcal{MC}$ where

$$\mathcal{MC} = \bigcup_{i=1}^{\infty} M^i$$

In other words, $\psi \in \mathcal{MC}$, is an arbitrary-length tuple where each element comes from a set M. M^k indicates the k-ary Cartesian product on the set M. $\mathcal{MC}$ is the union all such k-ary Cartesian products. In the prosody-level task, M is nothing but a set consisting of all the syllable weights, which happens to be a set of cardinality 2 for Sanskrit. $\mathcal{MC}$ consists of all the possible sequences of syllable weights, which are crucial

19 The morphological tag for the nominal is of the form *case—number—gender*.

for metrical patterns. In morphosyntactic tasks, M is defined such that an element in it is either a complete combination of categories, which leads to a valid morphological tag, or a partially complete combination of grammatical categories. We define the complete combination and partial combination as follows:

1. *Complete combination* (i.e., a morphological tag) – Any combination of grammatical categories that forms a valid morphological tag for some lexical category in the language is considered a complete combination. For instance, a nominal is represented by case, gender, and number. Hence, the combination genitive–masculine–singular forms a complete combination for the lexical category Noun.
2. *Partial combination* – A combination of grammatical categories, which can form a morphological class by adding one or more categories to it. For instance, genitive–masculine is a partial combination that denotes all the possible (three) complete combinations, which differ from each other only in terms of the category number. However, genitive–first person is not a valid combination as it can never form a valid morphological tag. The evidence for a partial combination in the corpus $\mathcal{C}$ can be obtained by summing the evidence of all the morphological classes that it can form.

We obtain a total of 528 different entries in set M, where we consider 240 complete combinations and 288 different partial grammatical category combinations. Here, we consider all complete and partial combinations of a noun, verb, and those denoted by "Others" in Table 2.

Thus, every feature is unique in two ways. One, given an edge, a feature captures only a partial information about the node pair the edge connects. This is because each feature contains only one attribute each from the nodes in the node pair connected by the edge. For instance, consider two ϵ-features, $f_1 = \langle a(1), \epsilon, a'(2)\rangle$, and $f_2 = \langle a(3), \epsilon, a'(1)\rangle$, for the edge connecting the tuples ($pituḥ, pitṛ, genitive|1|masculine$) and ($rāmaḥ, rāma, nominative|1|masculine$). Then f_1 calculates the feature value based on distributional information between $pituḥ$ and $rāma$, whereas f_2 calculates the feature value between $genitive|1|masculine$ and $rāmaḥ$. Two, it can additionally use a constraint ψ to limit the distributional context under which the association strength between $a(i)$ and $a'(j)$ should be measured. For instance, $\langle a(1), \psi, a'(2)\rangle$ is different from $\langle a(1), \epsilon, a'(2)\rangle$ and results in different feature values, as the former has an additional constraint ψ based on which the distributional information is calculated. The distributional information is calculated from a morphologically tagged corpus. We first construct a typed graph $\mathcal{C}$ from the corpus, henceforth to be referred to as the corpus graph. Every node in $\mathcal{C}$ has exactly one type. For morphosyntactic tasks, $\mathcal{C}$ has 530 types and the set of types is defined as $T = \{inflected\ form, stem\} \cup M$. This implies that the type of a node in $\mathcal{C}$ can either be a partial combination in M, a complete combination (morphological tag) in M, an inflected form or a stem. The number of nodes of the types *inflected form* and *stem* in $\mathcal{C}$ depends on the vocabulary of the corpus, as every unique stem and inflected-form will form nodes in $\mathcal{C}$. However, there will only be one node each in $\mathcal{C}$ for each type in M. Prosody-level task has a corpus graph $\mathcal{C}$ with just 3 types, the surface-form of the syllable and the two syllable weight values (i.e., "light" and "heavy"). The number of nodes here will depend on the number of unique syllables present in the corpus; and further, we have one node each for the types heavy and light. Based on the construction of $\mathcal{C}$, a feature $f = (a(i), \psi(1), \psi(2), \ldots\ldots \psi(k), a'(j))$ can be seen as a typed path in $\mathcal{C}$,

where each node in the typed path corresponds to a type in T. This translates to a horn clause $Edge(a(i), \psi(1)) \wedge Edge(\psi(1), \psi(2)) \ldots . \wedge Edge(\psi(k-1), \psi(k)) \wedge Edge(\psi(k), a'(j)) \rightarrow Edge(a(i), a'(j))$, where $Edge(x, y)$ is a predicate defined as "an edge from x to y." Thus, the features essentially denote different paths to reach from $a(i)$ to $a'(j)$ in $\mathcal{C}$, and the feature value for a given node pair will measure the strength of the path. Next, we discuss the feature value computation process using PCRW.

Feature Score. We use PCRW (Lao and Cohen 2010) for calculating the feature values. The feature values are calculated based on the distributional information from the corpus graph $\mathcal{C}$. Two nodes in $\mathcal{C}$ are connected, irrespective of their types, if they co-occur in a sentence in the underlying corpus. Further, for every node in $\mathcal{C}$, we calculate its frequency of occurrence in the corpus. Such a formulation for $\mathcal{C}$ enables us to use the PCRW for the calculation of feature values. For instance, consider an ϵ-feature, $\langle a(i), \epsilon, a'(j) \rangle$. Here, the attributes $a(i)$ and $a'(j)$ are nodes in $\mathcal{C}$ and because the feature has no additional constraints, it simply forms an edge between the nodes $a(i)$ and $a'(j)$ in $\mathcal{C}$. The score for an ϵ-feature translates to the co-occurrence probability of both the attributes in sentences across the corpus. This is defined as P_{CO}:

$$P_{CO}(a(i)|a'(j)) = \frac{count(a(i), a'(j))}{count(a'(j))}$$

Now, the feature score calculation in the presence of morphological constraints $\psi \in \mathcal{MC}$, can be formulated as follows. For a feature $\langle a(i), \psi, a'(j) \rangle$, where $\psi = (\psi_1, \psi_2, \ldots \psi_k)$:

$$P_{\psi}(a(i)|a'(j)) = P_{CO}(a(i)|\psi_1) \times \left(\prod_{i=2 \ldots k} P_{CO}(\psi_{i-1}|\psi_i) \right) \times P_{CO}(\psi_k|a'(j))$$

The feature essentially is a typed directed-path with multiple edges. The score for the typed path is the product of co-occurrence probability of the edges in the typed path. We next describe about FSPG, the approach we use for learning the feature function.

Learning of Feature Function. We use the FSPG Algorithm (Meng et al. 2015) for the learning of feature function. The FSPG framework essentially is an extension of Least Angle Regression approaches (LARS; Efron et al. 2004), which uses greedy strategies for feature subset selection from a potentially infinite feature space. The approach is used as a pretraining step, with an auxiliary task as Krishna et al. the objective for the supervised regressor. We first define a regression model, where the ground truth is obtained based on a measure of binary association between a pair of words in a corpus, such as bigram, co-occurrence probability, or pointwise mutual information (PMI; Krishna et al. 2018). The method iteratively chooses the most relevant feature that can distinguish the strong associations from the weak associations. The choice for the most relevant feature is made greedily, where the method chooses a feature with the strongest correlation to the expected output (Meng et al. 2015). For this purpose, a vector of residual values, $\vec{r}$, is constructed. The residual vector essentially is the element-wise difference between the values of the ground truth examples and the predictions from the current model, for the training data given as input (Meng et al. 2015). Now, for a new feature to be added we create a vector $\vec{m}$, where we calculate the feature score for the training data given as input. The cosine similarity between $\vec{r}$ and $\vec{m}$ for each of such features is calculated, and the feature with the highest cosine similarity is chosen as the

feature to be added. The algorithm terminates until the residual vector $\vec{r}$ is negligible (i.e., $|\vec{r}| < \epsilon$).

Meng et al. (2015) adapt LARS to handle infinite feature spaces by proposing the GreedyTree Algorithm (Meng et al. 2015, Algorithm 2), a greedy approach for finding relevant features iteratively. Similar in spirit to the approaches used in Monte Carlo Tree Search, the GreedyTree algorithm follows a best first search, which explores a graph by expanding the most promising node in the graph (Gaudel and Sebag 2010). A feature in our framework is defined as a tuple $(a(i), \psi(1), \psi(2), \ldots \psi(k), a'(j))$. This tuple is represented as a path in the GreedyTree. Here, each element of the tuple is a node. A path keeps expanding until a certain lower bound for a priority score is obtained. After a feature with the highest priority is expanded, the tree structure is preserved for subsequent feature generation passes (Meng et al. 2015). Krishna et al. (2018) perform exhaustive enumeration of the features and then perform feature selection using PRA. But in PRA the exhaustive enumeration was achieved in practice by limiting the maximum length of the feature tuple to an arbitrary length. In both PRA and FSPG, the feature score is calculated using PCRW.

Vector Formation in the Merged Representation for Prosody-Level Tasks. In Section 3.1, we discussed merging of nodes in "unambiguous paths" for graph construction in the prosody-level task. Here nodes are formed at a syllable level, but the nodes in unambiguous paths of a word are merged together to form a single node. For instance, in Figure 8(a), consider a path of two edges *ma* $\rightarrow$ *hā* $\rightarrow$ *ba*. In Figure 8(b), this was merged to form a single node, *mahāba*. The edges *ba* $\rightarrow$ *laḥ* and *ba* $\rightarrow$ *lo*, will now become *mahāba* $\rightarrow$ *laḥ* and *mahāba* $\rightarrow$ *lo*, respectively. Because the graph structure with the merged node forms the input to the structured prediction model, the edge vector generator has to generate the feature vectors for the edges in the graph with merged nodes. However, the corpus graph $\mathcal{C}$ for the prosody-level task contains syllable-level nodes only. To obtain the edge vector for an edge (e.g., *mahāba* $\rightarrow$ *lo*), we first obtain the edge vectors for all the edges in the original graph structure without any merged nodes. In this case we obtain three vectors, one each for the edges *ma* $\rightarrow$ *hā*, *hā* $\rightarrow$ *ba*, and *ba*$\rightarrow$ *lo*. We then form a new vector by performing element wise max operation using all the edge vectors in the path *ma* $\rightarrow$ *hā* $\rightarrow$ *ba* $\rightarrow$ *lo*. This operation is similar to the max-pooling operation. It implies that the new vector will contain the maximum value at each dimension of the vector among the three edge vectors in the path.

3.3 Inference Procedure

We use the inference procedure to find the output variable Y, the structure with the minimum energy, where Y will always be an induced subgraph of the observed variable X. However, the inference procedure is task specific and it determines the search space of the possible output structures from which the structure with the minimum energy is predicted. Table 4 mentions the different inference procedures our framework uses and the specific structures they predict. For instance, Algorithm 2 is used for the joint task of word-segmentation, morphological parsing, and dependency parsing. The procedure is basically a slightly modified version of Prim's algorithm, used here as an approximation algorithm to find directed Steiner trees (Takahashi 1980). Similarly, Algorithm 1 is used for the tasks of word segmentation, morphological parsing, and the joint task of word segmentation and morphological parsing (Krishna et al. 2018). Here, we start the clique selection with a single node. At any given instance, we loop through the nodes in the graph that are not yet part of the clique. We add a vertex v to the clique if the

cumulative score of all the edges from v to every vertex that is already in the clique is the minimum. We discard all the nodes that conflict with vertex v. As guaranteed by our structured input graph construction, we obtain the maximal clique (*exhaustive segmentation*) when there exist no more vertices to loop through. We perform this for every node in the graph X. From all the cliques so obtained we select the maximal clique with the least score. The approach does not guarantee enumeration of all the cliques, but it is guaranteed that every node will be covered by at least one maximal clique. These approximate algorithm-based inference procedures can be seen as a means of sampling some potential minimum energy maximal cliques for the learning task. EBMs do not require proper normalization of the solution space (LeCun et al. 2006), a choice that enables the use of the heuristic. During inference, the greedy clique selection heuristic is performed for every node in X. Though the run-time for this inference is polynomial, it can still be computationally expensive. In practice, we find that our inference procedure results in faster output for graphs with > 19 nodes in comparison to the exponential time Bron–Kerbosch algorithm (Tomita, Tanaka, and Takahashi 2006; Bron and Kerbosch 1973) for clique enumeration (McDonald et al. 2005a). We further improve the run time of our inference procedure by paralleling the clique selection procedure for each node on a separate thread.

We use Beam search as our inference for the linearization task both in standalone and joint settings, and for the prosodification task. In the joint word segmentation, morphological parsing, and syntactic linearization task, similar to the other joint tasks, only a subset of the nodes from the input X will be in the predicted structure. In such tasks, we encode the constraints in the inference such that once a node is selected, its "conflicting nodes" are removed from the search space. This can be observed in Step 4 of Algorithms 1 and 2 and this is used in the beam search inference for the Joint WS+MP+SL task as well. "Conflicting" nodes are any pair of nodes that are not connected by an edge between them. This follows from the construction of the graph X, as the non-connectivity between the nodes implies that they are proposed as alternative word suggestions in X. The removal of such nodes guarantees that the structures predicted by these inference procedures will always result in an "exhaustive segmentation." In fact, the inference procedures for all the tasks, with an exception to the dependency parsing task, are essentially approximation algorithms. For dependency parsing, we use the exact search-based inference using the Edmonds–Chu–Liu algorithm for obtaining the minimum energy-directed spanning tree (McDonald et al. 2005b).

Algorithm 1 Greedy maximal clique selection heuristic

1: **for** each node v_i in V_X **do**
2: Initialize a graph $K_i(V_{K_i}, E_{K_i})$ with $K_i = X$ such that $V_{K_i} = V_X$ and $E_{K_i} = E_X$. Initialize a vertex set V_{T_i} with v_i as the only element in it. Remove all the vertices that are conflicting with v_i from K_i.
3: Add the vertex $v_j \in (V_{K_i} - V_{T_i})$ to V_{T_i}, such that in K_i, the sum of edge weights for the edges starting from v_j to all other vertices in V_{T_i} is minimum.
4: Remove all the vertices that are conflicting with v_j from V_{K_i}.
5: Repeat steps 3–4 till $V_{K_i} - V_{T_i} = \varnothing$
6: **end for**

Algorithm 2 Approximation algorithm for finding directed Steiner Tree using Prim's algorithm originally proposed for Minimum Spanning Tree

1: **for** each node v_i in V_X **do**
2: Initialize a tree $T_i(V_{T_i}, E_{T_i})$ with v_i as the only vertex in it. Initialize a graph $K_i(V_{K_i}, E_{K_i})$ with $K_i = X$ such that $V_{K_i} = V_X$ and $E_{K_i} = E_X$. Remove all the vertices that are conflicting with v_i from K_i.
3: Find the minimum weighted directed edge in E_{K_i}, which has its source node in one of the nodes in V_{T_i} and target node in one of the nodes in $(V_{K_i} - V_{T_i})$. Add this edge to E_{T_i} and its target node v_j to V_{T_i}.
4: Remove all the vertexes conflicting with v_j from V_{K_i}.
5: Repeat Steps 3–4 till $V_{K_i} - V_{T_i} = \varnothing$
6: **end for**

3.4 Design Decisions

The configurations of the EBM we have discussed so far can be seen as language-agnostic. The language-specific components come from the graph generator, where we use SHR and MIT for the morphosyntactic and prosody-level tasks, respectively. The framework treats the input as a graph without incorporating any language-specific constraints. But, the architecture enables one to incorporate known properties about the language and then perform constrained optimization over the structure to be predicted (Belanger, Yang, and McCallum 2017). We experiment with EBM configurations where we incorporate such language-specific constraints for the dependency parsing and syntactic linearization. The linguistic information is made to use both in pruning the search space and in filtering the candidates during the inference. Essentially, these constraints can be seen as higher order features that capture constraints that span beyond a pair of words. For dependency parsing we rely on the linguistic information from rule-based dependency analyzers proposed for Sanskrit (Kulkarni, Pokar, and Shukl 2010; Kulkarni and Ramakrishnamacharyulu 2013; Kulkarni 2013). In the language-agnostic version of dependency parsing we form a complete graph as input. However, several of these edges might not be valid as per the traditional Sanskrit grammar. Such invalid edges can be determined and pruned using the linguistic information from the grammar, already used in the aforementioned rule-based dependency parsers. For several MRLs, including Sanskrit, morphological markers are indicative of the presence and also the type of syntactic dependency between the words in a sentence (Nichols 1986; Seeker and Kuhn 2013). Further, morphological markers may also be indicative of the agreement that needs to be fulfilled between the words in a syntactic relation (Nichols 1986). For instance, the subject and verb must agree on the "Number" grammatical category in Sanskrit.

In Sanskrit, case markers of a nominal are indicative of the syntactic relations it can have with a verb and in some case with other nominals. For instance, consider two candidate nodes (*ādesāt*, *ādesa*, *ablative|1|masculine*) and (*rāmaḥ*, *rāma*, *nominative|1|masculine)* in the graph X for the reference sentence. As these two candidate nodes are not conflicting, they will have two edges between them, one in either direction. However, as per the traditional Sanskrit grammar, a nominal in nominative case cannot form a valid syntactic relation with a nominal in ablative case. Hence, the edge can be

removed from X.[20] Summarily, we prune all those edges where no relation is possible between the words as per Sanskrit grammar. The linguistically informed pruning can happen not only prior to the inference, but also during the inference. Consider the words *rāmaḥ* and *sumukhaḥ* in the reference sentence. Both the nominals are in nominative case, and hence both are eligible to have the verb *aicchat* as their head for the *kartā* (subject) relation. The linguistic pruning approach cannot distinguish one from the other, as both have the same morphological information, and will keep both the edges in the input. However, during inference, once one of these two edges is chosen as part of the (partial) predicted structure, then the other edge can be pruned from the search space. This is because the verb *aicchat* can form an edge with at most one nominal that is in nominative case.[21] We also incorporate knowledge that lies outside of morphological markers, such as roles of specific indeclinable words, as well as cases where an indeclinable acts like a noun by taking a declension and forming noun–noun relations (Kulkarni, Pokar, and Shukl 2010, §2.2). We add numerous constraints that mandate or invalidate the presence of certain word-pair connections and relations in the final solution, by virtue of presence of one or more relations in the solution (Kulkarni and Ramakrishnamacharyulu 2013, §4). Such constraints are actively used in the linguistic tradition of dependency analysis in Sanskrit (Kulkarni, Pokar, and Shukl 2010, §2.1). We altogether incorporate about 120 constraints, which include rules and exceptions that make use of morphological and morphosyntactic features, lexical lists, and lists of indeclinables, and so forth.

We incorporate additional linguistic information for syntactic linearization as well (Kulkarni et al. 2015; Scharf et al. 2015). Similar to the dependency parsing task, we make use of the rule-based constraints for dependency analysis (Kulkarni, Pokar, and Shukl 2010) to obtain a pruned search space. Further, on the pruned search space we apply the principles of dependency locality (Gibson 1998; Gibson et al. 2013) and weak non-projectivity (Kulkarni et al. 2015, §3.2 and §4.3) to filter out invalid partial candidate sequences. We perform each of these during inference and apply these principles on partially predicted sequences. The three steps we use are: (1) Take candidate (partial) sequences from the beam, and evaluate if a dependency tree can be made using the rule-based parser from Kulkarni (2013).[22] (2) Based on the tree construction, we check for violation of weak non-projectivity and minimum dependency length. (3) If any of the sequence is more than double the minimum dependency length (Gildea and Temperley 2010) and has more than 2 weak non-projectivity violations, then we discard the candidate. For the partial sequence we always maintained a dummy root, to which all the dangling nodes were connected to form a tree.[23]

4. Experiments

We present the results for experiments for all the tasks that we have described so far. We first look into the performance of standalone tasks and compare them with other SoTA baselines whenever possible and show the effectiveness of our model. Then we present

20 A possible exception to this rule is when one of the nominals is derived from a verb. In such cases, it is possible that the nominal may act as a verb. In such a case, we do not prune the edge.

21 A possible exception to this might be for those verbs like *gamayati*, which have a causative marker in it. As SHR is equipped with this analysis as well, we consider such cases as well.

22 While we used the same set of rules from Kulkarni (2013), we re-implement the parser.

23 The number for minimum dependency length and number of violations of weak non-projectivity were decided empirically on the development data.

the results for the joint formulation of the tasks and compare them to the pipeline-based approaches. We further evaluate settings where linguistic characteristics specific to Sanskrit are taken into consideration. Finally, to show the language-agnostic nature of the framework, we present the results for Czech. We train two standalone models for dependency parsing and morphological parsing and one model for joint MP+DP.

4.1 Data Set

We use the Digital Corpus of Sanskrit (DCS) (Hellwig 2010–2016), a morphologically tagged corpus of Sanskrit, for all our experiments. The corpus mostly consists of texts from epics and scientific domains such as medicine, astronomy, grammar, and so on (Hellwig and Nehrdich 2018). DCS contains digitized works from periods that span over 3,000 years, and includes works written in prose, poetry, or a mix of both. This makes the corpus a collection of texts that are stylistically, topically, and chronologically diverse. It contains more than 66,000 unique lemmas and 3,200,000 tokens. DCS currently consists of 561,596 lines of text, where every line has undergone word segmentation and morphological parsing. We use a subset of 350,000 textlines from DCS for the construction of the corpus graph. We mostly reuse the subset of DCS containing 350,000 textlines, originally used by Krishna et al. (2016) for the word-segmentation task and later in Krishna et al. (2018) for the joint WS+MP task. Some of these textlines were removed from the original subset as they formed part of the train, test, or dev data set for the newly introduced tasks in this work. However, to maintain the count at 350,000, new textlines, sampled randomly from DCS, were added. The train, test, and development split for each of the tasks discussed in this work is shown in Table 5.

Word Segmentation and Morphological Parsing Krishna et al. (2018) used a training set of 8,200 textlines from DCS for training the joint model for word segmentation and morphological parsing. In Krishna et al. (2018), the segmentation results were reported on a data set of 4,200 sentences (termed as DCS4k), and results for morphological parsing were reported on a data set of 9,576 sentences called the DCS10k. In this work, we report the results of morphological parsing on the same DCS10k data set. However, for word segmentation we use a subset of the DCS10k with 8,925 textlines, as about 650 textlines were either missing or had alignment issues with the data set used for one of the baselines (Hellwig and Nehrdich 2018).

Table 5
Number of textlines used as the train, development, and test splits for the tasks in this work.

Task	Train	Dev	Test
WS	8,200	2,000	8,925
MP	8,200	2,000	9,576
WS+MP	8,200	2,000	9,576
DP	12,320	2,000	1,300
WS+MP+DP	12,320	2,000	1,300
SL	10,000	2,000	3,017
WS+MP+SL	10,000	2,000	3,017
Prosodification	10,000	2,000	3,017

Dependency Parsing DCS is a morphologically tagged corpus, but does not contain annotations for dependency parsing. We obtain 4,000 dependency tagged sentences from the Sanskrit Treebank Corpus (Kulkarni 2013) and the corpus of Śiśupālavadha.[24] From the collection, we choose a subset of 1,300 sentences as our test data set. For training, we do not use any gold standard sentences directly, but obtain 12,320 sentences by augmenting the remaining 2,700 gold standard sentences. Dependency parsing is the only task where we do not use any gold standard data for training. Given the restricted settings under which we train the model, we make sure that every word in the augmented data and the test data (from the original data) has its stem present in the corpus graph $\mathcal{C}$. However, 13.81% of the inflected forms are out of vocabulary in $\mathcal{C}$. For obtaining the 12,320 augmented sentences, we relied on standard data augmentation approaches followed generally in text classification, semantic role labeling, and dependency parsing. In particular, we perform synonym replacement[25] of at most one word per sentence (Zhang, Zhao, and LeCun 2015), such that the stem for a new word is found in DCS. We additionally perform sentence simplifications (Vickrey and Koller 2008), originally proposed for SRL, and sentence cropping approaches (Sahin and Steedman 2018) proposed for dependency parsing. In sentence cropping, we make sure that every cropped sentence still maintains at least one subject, object, and verb and we primarily target cropping of non subject/object words (and their subtrees) in the sentences. While augmentation may lead to sentence-level semantic shifts, it will still preserve local syntactic tags (Sahin and Steedman 2018). We altogether obtain a set of 20,000 augmented sentences. We further apply linguistic constraints based on the principles of verbal cognition (Bhatta 1990) and Kāraka (Sanskrit-specific dependency analysis) (Kulkarni and Ramakrishnamacharyulu 2013) and filter 12,320 sentences from the 20,000 augmented sentences. We use the filtered set of sentences as our training data, and each sentence has an average length of 6.42 words per sentence. This is comparable to the average number of words in a sentence in the test data, which is 7.03. The test data is devoid of any augmentations, and is in its original form. We also filter 2,000 sentences from the remaining sentences (7,680 of 20,000) to use as development data.

Syntactic Linearization and Prosodification We obtain 17,017 pair of verses and their corresponding *anvaya* from the epic Rāmāyaṇa from Śrīnivāsa Aiyaṅkār (1910).[26] Krishna et al. (2019), the current state of the art in syntactic linearization in Sanskrit, use a test set of 3,017 textlines from this data set. We use the same test data for all these three tasks. From the remaining data, we use the 10,000 textlines as training data. Standalone syntactic linearization takes a BoW as input and predicts the *anvaya*. Prosodic linearization takes a BoW as input and predicts the verse. Joint WS+MP+SL takes a verse as input and predicts the *anvaya*.[27] We compare our EBM configurations with the SoTA neural baselines for the standalone tasks. These baselines are provided with additional data for their training. For syntactic linearization, we use 95,000 prose order sentences crawled from Wikipedia by Krishna et al. (2019). Similarly for prosodification, we use 100,000 verses from DCS and Vedabase.[28]

24 `http://sanskrit.uohyd.ac.in/scl/e-readers/shishu/#/home`.

25 We use the lexical networks Indowordnet (Kulkarni et al. 2010) and Amarakośa (Nair and Kulkarni 2010).

26 The 17,017 verse-anvaya pairs are filtered from 18,250 verse-anvaya pairs available at `http://bit.ly/ramayanapairs`. The rest of them are ignored due to word-level mis-alignments between verses and their corresponding *anvaya*.

27 The input, i.e., verse, will retain the original written form, including fused word-forms due to *sandhi*. The *anvaya* will be a sequence of segmented words.

28 `https://vedabase.io/en/`.

4.2 Baselines

Table 6 shows the list of baseline systems and the various configurations of our EBM framework, along with the tasks for which these systems are used. We provide a brief description of each of the baseline systems.

4.2.1 Word Segmentation.

SupervisedPCRW (Sup-PCRW) Krishna et al. (2016). Similar to the EBM-framework, the model uses the graph output from SHR. It treats the problem as an iterative query expansion problem, where a greedy heuristic approach is used to select candidate nodes

Table 6
Different baseline systems and configurations of EBM used in various tasks. The tick mark (✓) indicates that the system is used for the task as a baseline. Star (✻) indicates that the system reports the best score for the task. The naming for EBM configurations is as follows: The inference procedure used, the term "EBM," followed by the type of the edge vector generation marked by its first character (PRA/FSPG). Here "Tree-EBM*-F" and "Beam-EBM*-F" are the two EBM configurations with language-specific augmentations as described in Section 3.4.

System	Tasks							
	WS	MP	DP	SL	WS+MP	WS+MP+DP	WS+MP+SL	PL
EG-CRF	✓	✓			✓			
Sup-PCRW (Krishna et al. 2016)	✓							
rcNN-SS (Hellwig and Nehrdich 2018)	✓							
FCRF (Malaviya, Gormley, and Neubig 2018)		✓						
SeqGen (Tkachenko and Sirts 2018)		✓						
NeurDP (Dozat and Manning 2017)			✓					
YAP (More et al. 2019)			✓					
EasyFirst (Goldberg and Elhadad 2010)			✓					
LinLSTM (Schmaltz, Rush, and Shieber 2016)				✓				✓
BSO (Wiseman and Rush 2016)				✓				✓
kāvya guru (Krishna et al. 2019)				✓				✓
Energy-Based Model (EBM) Configurations								
VL-EBM	✓	✓			✓			
BL-EBM	✓	✓			✓			
Prim-EBM-P	✓	✓			✓	✓		
Prim-EBM-F						✻		
Tree-EBM-P			✓					
Tree-EBM-F			✓					
Tree-EBM*-F			✻					
Cliq-EBM-P	✓	✓			✓			
Cliq-EBM-F	✻	✻			✻			
Beam-EBM-F				✓			✻	✻
Beam-EBM*-F				✻				
ATSP-EBM-F				✓			✓	✓

based on the edge weights. The edges are featurized with hand-crafted morphological features. The feature values are calculated using PCRW (Lao and Cohen 2010),
EdgeGraphCRF (EG-CRF). This is a second order CRF model (Müller and Behnke 2014; Ishikawa 2011) that also uses the SHR output graph X as the input to the system. Every node is represented with fastText (Bojanowski et al. 2017) word embeddings, trained with inflected forms of the words from the combined data set of Bhāgavatam from Vedabase and the DCS data set (Hellwig 2010–2016).[29] The edges are featurized with the PRA vectors. We used 1-slack structured SVM for training. QPBO (Rother et al. 2007) inference approach provided the best results for the model.
Char RNN-CNN Seq-Tagger (rcNN-SS). Hellwig and Nehrdich (2018) propose a sequence tagger that uses character-level recurrences and convolutions to train a Sanskrit word segmenter. The best performing model of theirs passes characters of the sequence through a bidirectional RNN and then performs convolution operations on the resulting embedding. Additionally, shortcut connections (Bishop 1995) are applied both from the input character embeddings and the RNN output to the resulting embeddings.
Lattice-EBM. This is an energy-based sequence labeling model, where the input is a lattice (Wolf and Woods 1977) similar to that of Kudo (2006). The model can be seen as a special case of Graph Transformer Networks (LeCun et al. 1998; LeCun et al. 2007). In the lattice structure, the candidate links only to its adjacent nodes in an exhaustive segmentation. We also generate edge vectors for the dummy nodes that act as the start and end markers in the lattice. We use the PRA vectors as the edge vectors for the model. During prediction, we have to find the best path from the lattice that minimizes the sentence score. Here, we consider two variants of Lattice-EBM. *VL-EBM* uses the discriminative forward training approach (Collobert et al. 2011) with the standard hinge loss. The second variant *BL-EBM* uses multimargin loss (Edunov et al. 2018) instead of the hinge loss. Here, we employ beam search to generate multiple candidates as required by the loss.
Prim-EBM-P. This is also a configuration of the EBM, where the model uses the input graph X from SHR, PRA vectors for the edges, and modified Prim's algorithm as the inference. The inference procedure searches for a Steiner Tree (Takahashi 1980) from the input graph X. Prim's algorithm acts as an approximation algorithm to find the Directed Steiner tree (Voss 1993). The Steiner tree essentially spans over a subset of nodes, and by graph construction it is guaranteed that the inference procedure will produce an exhaustive segmentation.
Clique-EBM (Cliq-EBM-P/Cliq-EBM-F). This is the proposed EBM configuration for the word segmentation (and morphological parsing) problem. Similar to other Prim-EBM-P, the configuration uses the input X obtained from SHR. Here we use our greedy maximal clique selection heuristic, as shown in Algorithm 1, as the inference procedure. In Krishna et al. (2018) the configuration used PRA edge vectors, and is denoted as Cliq-EBM-P. Here, we additionally introduce a variant Cliq-EBM-F that uses FSPG-based edge vectors for the task. The model works exactly the same as Prim-EBM-P but differs only in the inference procedure used.

4.2.2 Morphological Parsing.

Neural Factorial CRF (FCRF). Malaviya, Gormley, and Neubig (2018) proposed a Factorial CRF model (Sutton, McCallum, and Rohanimanesh 2007) that relies on an LSTM-based

29 `https://vedabase.io/en/library/sb/`.

sequence model for its feature representation. The model predicts a composite label for each word in the sequence, where a label is a set of tags, one for each of the grammatical category. Here, co-temporal factors are defined to capture the dependencies between various tags of the same word. To capture the dependencies across the words, there exist factors between tags of the same type for the adjacent words, thereby forming a linear chain.

Sequence Generation Model (SeqGen). The model, proposed in Tkachenko and Sirts (2018), also treats the label as a composite label. Here, a char-BiLSTM is used to obtain word-embeddings, which are then passed on to a word-level BiLSTM as the input features (Lample et al. 2016; Heigold, Neumann, and van Genabith 2017). The model, inspired from seq2seq models, uses an LSTM decoder to predict the grammatical categories one after the other, based on the previous predictions for each word.

Energy-Based and CRF-Based Configurations. All the configurations of the EBMs, in addition to Edge Graph CRF, which is used for the word segmentation task (Section 4.2.1), are used in the morphological parsing task as well. These models predict all the morphemes of a given word, including the stem of a word. But the aforementioned neural baselines only predict the morphological label of a word and not the stem. In these models, we use segmented sentences obtained from the gold standard data in DCS, as input to the SHR. SHR does not analyze for *sandhi* and only performs morphological analysis of the words involved. This forms the input graph X for these models.

4.2.3 Dependency Parsing.

Yet Another Parser (YAP). The model is a language-agnostic dependency parser for morphologically rich languages (More et al. 2019). The transition-based framework currently reports the best score for dependency parsing and for the joint task of morphological analysis and dependency parsing in Hebrew. Further, the framework was experimented on multiple languages for dependency parsing (Seker, More, and Tsarfaty 2018).

Easy-First Parser (EasyFirst). The model proposed by Goldberg and Elhadad (2010) is a non-directional greedy search procedure used for dependency parsing. The parsing procedure is initialized with all the words in a sentence. The system iteratively forms partial structures by merging existing partial structures, where only the head partial structure is retained in the list of structures available for merging. The procedure terminates when there is exactly one structure remaining, which corresponds to the root of the sentence.

Neural Graph-Based Dependency Parsers (NeurDP). We experiment with two neural dependency parsers, the deep biaffine attention-based model of Dozat and Manning (2017) and the neural graph-based dependency parser based on Kiperwasser and Goldberg (2016a). Both the models rely on LSTM-based feature extractors for the parsing.

EBM Configurations. Our EBM configurations are basically graph-based parsing framework for dependency parsing. Here, similar to McDonald et al. (2005b), we use Edmond-Chu-Liu's algorithm for finding arborescence (directed spanning tree) of minimum weight.[30] We experiment with three variations: Tree-EBM-P, Tree-EBM-F, and Tree-EBM*-F. Tree-EBM-P uses PRA vectors as the edge vectors, and Tree-EBM-F uses the edge vectors using FSPG. Here, Tree-EBM*-F incorporates the language-specific

30 While McDonald et al. (2005b) used the approach to find the maximum weighted arborescence, we use this to find the minimum weighted arborescence.

augmentations as discussed in section 3.4. For all our EBM models, the label prediction is fully integrated with our parsing procedure, similar to Kiperwasser and Goldberg (2016b).

4.2.4 Syntactic Linearization and Prosodification.

LSTM-Based Linearization Model (LinLSTM). The model (Schmaltz, Rush, and Shieber 2016) essentially is a neural language model (Bengio et al. 2003) with a beam search–based decoder. First, a language model is learned by feeding an LSTM-based neural network with ground truth sequences, namely, sequences with desired word order. Beam search is used for decoding from the language model. We use a beam size of 512, as reported in Schmaltz, Rush, and Shieber (2016), for all our experiments. This model currently reports the best result in the word ordering task for English (Hasler et al. 2017), even outperforming other syntax-based linearization models.
Seq2Seq Beam Search Optimization (BSO). This is a seq2seq model (Wiseman and Rush 2016), where the training procedure defines a loss function in terms of errors made during beam search. The beam always keeps multiple top-*k* possible sequence hypotheses as possible solutions. The loss penalizes the function when the gold sequence falls off the beam during training. The model during inference is constrained to predict only from the words that are present in the input at the encoder side. Additionally, the model is further constrained to predict only from the entries in the input that are yet to be predicted. BSO has been shown to outperform LinLSTM by a huge margin, in English, when both the models used the same beam size for decoding (Wiseman and Rush 2016). We use a beam size of 15 and 14 for testing and training, respectively, the setting with the best reported scores in Wiseman and Rush (2016).
kāvya guru. This is a seq2seq model (Krishna et al. 2019) with gated CNNs (Gehring et al. 2017), using a sequence-level loss (Edunov et al. 2018). The model uses a weighted combination of expected risk minimization and the token-level log likelihood with label smoothing as loss. Additionally, the model uses two pretraining components: One for learning task-specific embeddings, formed by a weighted combination of pretrained word embeddings (Kiela, Wang, and Cho 2018); and, second, a component for generating multiple ordering hypotheses (Wang, Chang, and Mansur 2018) to be used as input to the seq2seq component. The model currently reports the state-of-the-art results for syntactic linearization in Sanskrit.
EBM-Based Configurations (Beam-EBM-F/ATSP-EBM-F). Here, for both the syntax-level and prosody-level linearization tasks, we use the exact same EBM configurations. The only difference will be in the input graph representation. For the tasks, we experimented with two different inference procedures. Horvat and Byrne (2014) proposed to solve the task of linearization as that of solving the Asymmetric Traveling Salesman Problem (ATSP) over a graph. We incorporate the same as our inference procedure in the ATSP-EBM-F configuration. Inspired from the works of Schmaltz, Rush, and Shieber (2016) along with Wiseman and Rush (2016), we use beam search as the approximate inference in the Beam-EBM-F procedure. We also report the performance score for Beam-EBM*-F, which incorporates the language-specific augmentations as discussed in Section 3.4. This configuration is used only for the syntactic linearization task.

4.2.5 Edge Vector Generation.

Path Ranking Algorithm (PRA). In Krishna et al. (2018), the authors use the PRA (Lao and Cohen 2010) for learning the feature function. PRA is a 2-step pipeline approach,

where the features are first exhaustively enumerated, and then filtered in the next step. In PRA, it is possible to have a morphological constraint tuple ψ of any cardinality. However, because an exhaustive enumeration of all possible features is required in PRA, the cardinality of the constraint tuples is typically restricted by an arbitrary upper bound. Krishna et al. (2018) used two different settings, one where the upper bound was set to 1 (i.e., $|\psi| \leq 1$), and another where the upper bound was set to 2 (i.e., $|\psi| \leq 2$). The feature selection was performed using recursive feature elimination (Kohavi and John 1997; Guyon et al. 2002, RFE) and Mutual Information Regression (Kraskov, Stögbauer, and Grassberger 2004, MIR). These supervised approaches used PMI and word-pair co-occurrence probability (Co-Occ) as the labels for the feature selection. Altogether, this resulted in eight different configurations of feature sets using PRA. Of the eight, the one that used MIR with co-occurrence probability as the label on a set of features with an upper bound of $|\psi| \leq 1$ reported the best performance. We use this as our default configuration for the PRA. The PRA-based edge vectors are generated only for morphosyntactic tasks; hence all the aforementioned settings are valid only for edge vectors used in morphosyntactic tasks.

Forward Stagewise Path Generation (FSPG). This is the feature learning approach we elaborated in Section 3.2. It does not require exhaustive enumeration of features and uses LARS for feature selection. Word-pair co-occurrence probability is used as the supervised label for LARS.

4.3 Results

Hyperparameters. We report all our experiments based on the hyperparameter settings shown in Table 7. We use separate hyperparameter settings for prosodification, and a common hyperparameter setting was used for all the morphosyntactic tasks. Hyperparameter tuning for morphosyntactic tasks was performed originally on the word segmentation task. We perform random search for the hyperparameter tuning and perform 100 iterations each for morphosyntactic and prosodic tasks. The hidden layer size was varied at a step size of 50—that is,$\{50, 100, ...800, 850\}$ for morphosyntactic tasks and $\{50, 100, ...500\}$ for prosodification. Margin loss discount, a constant between 0 and 1 multiplied to the margin, and dropout were varied at a step size of 0.1. Similarly, the learning rate was varied in multiples of 0.1 $\{1 \times 10^{-01}, 1 \times 10^{-02}, ...1 \times 10^{-07}\}$. Additionally for each of the morphosyntactic tasks, we change the values of one hyperparameter at a time and observe if that leads to any improvement in the results, from the ones reported in Tables 8–15 for those tasks. We find that increasing the margin loss discount leads to marginal improvements for the syntactic linearization task, and leads to a 1.2 BLEU improvement when the margin loss discount is 1, that is, when no discount is applied. Keeping the hidden layer size to 750 leads to a 0.5 UAS and 0.4 LAS improvement for Tree-EBM*-F for dependency parsing, though no particular trend could be observed in varying the hidden layer size. Similarly, setting the dropout to 0.2 leads to an improvement 0.9 F-Score for the joint WS+MP+DP task.

Evaluation Metrics. For the standalone tasks, we report the standard metrics. For dependency parsing, we use UAS and LAS. Similarly for linearization tasks, we follow Krishna et al. (2019) and report the performance of the systems using BLEU (Papineni et al. 2002), Kendall's Tau (τ) score (Lapata 2003), and perfect match score, namely, the percentage of sentences with exact match to the input. For WS, MP, and the joint task of WS and MP, we use macro-averaged Precision, Recall, and F-Score (Krishna et al. 2018). We adopt the same metric for the joint task of WS, MP, and dependency parsing. As

Table 7
Hyperparameter settings for all the EBM models, based on which the results in Tables 8–15 are reported. The hyper parameter settings are tuned for the task word segmentation.

System	Morphosyntactic Tasks	Prosodic Tasks
Hidden Layer Size	650	400
Learning rate	1×10^{-05}	1×10^{-03}
Margin Loss Discount	0.3	1
Dropout	0.4	0.5

More et al. (2019) previously noted, the use of standard metrics for dependency parsing (i.e., UAS and LAS) need not be feasible in the joint setting due to the possibility of structural mismatch between the prediction and the ground-truth. For the same reason, we do not use the Kendall's Tau score for reporting the performance of the joint task of WS, MP, and syntactic linearization. Instead, we use the longest common subsequence ratio for this joint task, along with BLEU and perfect match scores.

Tables 8–12 show the results for each of the five standalone tasks as discussed in Section 2.1. In all the tasks, we can find that our models, using the EBM framework, report the best performance for the tasks. We elaborate the results for each of the tasks and compare the system performance with the baseline models.

Word Segmentation. Table 8 shows the results for the word segmentation task. The proposed system, Cliq-EBM-F, reports the best score in terms of macro average Precision, Recall, and F-Score for the task. Here, rcNN-SS (Hellwig and Nehrdich 2018) scores the best when it comes to perfect match score. Cliq-EBM-F reports a percentage improvement of 1.15% over rcNN-SS in F-Score.[31] The system named as Cliq-EBM-P was the system that was proposed in Krishna et al. (2018). Previously, the word-segmentation systems rcNN-SS and Cliq-EBM-P showed that they both outperform the then state of the art, a seq2seq model from Reddy et al. (2018). But a comparison between both these systems was not available. Here, we find that the rcNN-SS outperforms Cliq-EBM-P with a percentage improvement of 1.82% in terms of F-score (Section 4.3). But, both Cliq-EBM-P and the best performing model Cliq-EBM-F were trained on 8,200 sentences, which is just about 1.5% of the training data ($>$ 0.55 million) used in rcNN-SS.

All the competing systems for word segmentation except for rcNN-SS use the linguistically refined output from SHR as their search space to predict the final solution. EG-CRF was trained on 10,000 sentences, whereas V-L-EBM and B-L-EBM were trained on 9,000 sentences, after which the models became saturated. The Prim-EBM-P, Cliq-EBM-P/F all use a training data of 8,200 sentences. Being a second-order CRF model (Ishikawa 2011), EdgeGraphCRF does not take the entire sentence context into account. Also, the QBPO inference does not guarantee prediction of exhaustive segmentation. In fact, 85.16% of the sentences predicted by the model do not correspond to an "*exhaustive segmentation*." Prediction of an *exhaustive segmentation* is guaranteed in all our EBM

31 Following Dror et al. (2018), we perform pairwise t-tests between the reported scores for these systems, and find that the scores are statistically significant ($p < 0.05$).

Table 8
Word segmentation results.

System	P	R	F	PM (%)
Sup-PCRW	73.37	82.84	77.82	31.32
EG-CRF	77.66	79.81	78.72	40.46
VL-EBM	84.11	86.94	85.5	56.52
BL-EBM	87.12	87.37	87.24	63.17
Prim-EBM-P	88.79	91.37	90.06	64.22
Cliq-EBM-P	94.48	96.52	95.49	81.57
rcNN-SS	97.15	97.41	97.23	**87.75**
rcNN-SS-350k	96.83	97.07	96.9	86.59
Cliq-EBM-F	**98.04**	**98.67**	**98.35**	85.25

Table 9
Morphological parsing results.

System	P	R	F
SeqGen	81.79	81.79	81.79
FCRF	80.26	80.26	80.26
EG-CRF	79.48	82.4	80.91
VL-EBM	79.67	79.67	79.67
BL-EBM	81.42	81.42	81.42
Prim-EBM-P	85.11	85.11	85.11
Cliq-EBM-P	93.52	93.52	93.52
Cliq-EBM-F	**95.33**	**95.33**	**95.33**

models (also in supervised PCRW) by virtue of the inference procedure we use. Cliq-EBM-P and Cliq-EBM-F differ only in the vector generation module used. Between these two configurations, the use of FSPG-based vectors result in a percentage improvement of about 3% in the results (Section 4.3). In the lattice-based configurations, V-L-EBM-P and B-L-EBM-P, the edges are formed only to position-wise immediate nodes in an exhaustive segmentation. Also the inference procedures process the input sequentially, from left-to-right. The approach substantially reduces the performance of the system. This justifies the additional cost incurred due to use of our non-sequential inference procedure. Similarly, Prim-EBM-P performs substantially worse as compared to Cliq-EBM-P. Here, the latter considers all the pairwise potentials between the inputs, while the former does not.

Morphological Parsing. Table 9 shows the results for morphological parsing in Sanskrit. We find that similar to word segmentation, our proposed EBM configuration, Cliq-EBM-F provides the state-of-the-art results for morphological parsing. Currently, there exists no morphological parser that performs both analysis and disambiguation of morphemes in Sanskrit, leaving aside the Cliq-EBM-P configuration reported in Krishna et al. (2018). Instead, as baselines, we utilize two widely used neural sequence taggers that reported state-of-the-art results on multiple morphologically rich languages, the FCRF (Malaviya, Gormley, and Neubig 2018) and SeqGen (Tkachenko and Sirts 2018). But they predict only the morphological tag of each word. For all the other models we predict all the morphemes of a word including its stem. All the EBM configurations we use for morphological parsing are used for the word-segmentation task as well.

Among the models with a sequential processing of input, SeqGen reports the best F-Score of 81.79. Our lattice-based EBM configuration reports an F-Score of 81.42%, close to SeqGen. Cliq-EBM-F, our best performing model, reports a percentage improvement of 16.55% over FCRF. All the EBM configurations with graph-based input representation and a non-sequential inference procedure in general report a score that is significantly greater than the sequential models. Further, Cliq-EBM-F reports a percentage improvement of about 2% in comparison to Cliq-EBM-P. All our EBM configurations used the same amount of training data as used in word segmentation, which is less than 10,000. At the same time, the neural sequence taggers were trained on a data set of

50,000 sentences. For morphological parsing in standalone setting, the word segmented sequence is used as input. Due to this, the system's precision will be the same as its recall for all the systems, except for EG-CRF. For EG-CRF, the QBPO inference it uses does not guarantee prediction of an exhaustive segmentation, and hence the number of predictions need not match with the number of words in the ground truth. Hence, the precision and recall can be different.

Dependency Parsing. Table 12 shows the results for dependency parsing in Sanskrit. Here, our energy-based model configurations outperform the other baseline models for the task. Among the EBM configurations, Tree-EBM*-F reports the best score of 85.32 UAS and 83.9 LAS. Tree-EBM*-F is an EBM configuration where we incorporate language-specific knowledge as constraints, as discussed in Section 3.4. We find that the configuration reports percentage improvement of 3.23% from the reported performance of Tree-EBM-F (UAS of 82.65). Further, we find that the performance improvements in LAS are much more substantial in Tree-EBM*-F, with 5.86%, as compared to LAS of 79.28 of Tree-EBM-F. Thus, incorporating linguistic constraints in Tree-EBM*-F helped much more in disambiguating between the edge labels. The highest gain, in absolute numbers, is shown by *karma* (object) relation. Tree-EBM*-F gets 1,076 cases of 1,153 cases correctly, both in terms of predicting the correct head and label, while Tree-EBM-F only has 861 cases correct. Tree-EBM-F had 292 incorrect predictions for *karma* relation; in 62 of these cases, a dependent node was assigned an incorrect head with which no relation is possible in Sanskrit. Such cases would get pruned prior to inference in Tree-EBM*-F, due to the linguistic information. Four relations, *apadāna*, *prayojanam*, *tādarthyam*, and *sampradānam*, had less than 50% recall in Tree-EBM-F, while Tree-EBM*-F has a recall greater than 50% for all the 22 relations. The lowest recall in Tree-EBM*-F is 56% for *tādarthyam*. It happens to be the lowest occurring relation with just 25 occurrences, and had a recall of just 24% in Tree-EBM-F. Among the aforementioned four relations, *prayojanam* has the highest occurrence (130), of which the number of correct predictions increased from 54 to 82 (63.08%).

YAP (More et al. 2019) reports an UAS of 76.69 and LAS of 73.02, lower than the least performing EBM configuration, Tree-EBM-P. YAP currently reports the state-of-the-art score for dependency parsing in Hebrew, outperforming previous models including that of Seeker and Çetinoğlu (2015). It may not be fair to compare YAP with Tree-EBM*-F because of the latter's advantage in explicit linguistic information. But as a fair comparison of two language-agnostic dependency parsing models, Tree-EBM-F reports a performance improvement of about 7.77% as compared to YAP. We experimented with neural models for dependency parsing such as those of Dozat and Manning (2017) and Kiperwasser and Goldberg (2016a), but both the models had UAS/LAS scores below 50. All of the models used a training data set of 12,000 sentences, as described in Section 4.1. Given that the neural models rely on the network architecture for automatic feature extraction, we hypothesize that the lack of sufficient gold standard training data may have resulted in their poor performance.

Syntactic Linearization. Table 10 shows the results for syntactic linearization. Here, our model Beam-EBM*-F, a model that incorporates additional linguistic knowledge about Sanskrit syntax, performs the best in comparison to the other models. The model improves the performance by more than 4 BLEU points (percentage improvement of 7.45%), in comparison to *kāvya guru*, the current state of the art for linearization in Sanskrit. Altogether, we report performance of three different configurations of EBM for the task, of which *kāvya guru* outperforms two of the configurations. But all the

Table 10
Results for syntactic linearization.

System	τ	BLEU	PM (%)
LinLSTM	61.47	35.51	8.22
BSO	65.38	41.22	12.97
ATSP-EBM-F	72.79	48.03	20.18
Beam-EBM-F	73.62	51.81	25.16
Kāvya Guru	75.58	55.26	26.08
Beam-EBM*-F	**78.22**	**59.38**	**27.87**

Table 11
Results for prosodification.

System	τ	BLEU	PM (%)
LinLSTM	42.96	15.42	5.18
BSO	48.28	20.12	6.8
Kāvyaguru	58.27	36.49	12.59
ATSP-EBM-F	63.24	42.86	17.67
Beam-EBM-F	**70.4**	**46.88**	**21.16**

neural sequence generation models including *kāvya guru* were trained with an additional training data set of 95,000 from Wikipedia amounting to a total of 108,000 training sentences. Our EBM models were trained only on 10,000 sentences. LinLSTM performed worse when additional data from Wikipedia was used. Hence we report the score of linLSTM where the additional data was not used. We experiment with two different inference procedures, ATSP-EBM-F and Beam-EBM-F. We find that the beam search inference with a beam size of 128 works the best for the task. Larger beams result in diminishing returns, with improvements not being significant but incurring computational overheads at the same time.

Prosodification. Table 11 shows the results for prosodification. Here, we find significant difference between both the EBM configurations presented and other baselines. *kāvya guru* reports the best score among the non-EBM configurations. But for Beam-EBM-F, the score jumps by about 10 BLEU as compared to *kāvya guru*. Similarly, *kāvya guru* performs significantly better than other neural sequence generation models. We observe that the improvement appeared only after we incorporated prosody information, that is, explicitly incorporating syllable weight information and then filtering the sequence that adheres to valid patterns of syllable weights. This was incorporated in one of the pretraining steps used to generate multiple hypotheses for the final seq2seq component of *kāvya guru*. The BLEU score improved from 23.16 to 36.49 with this additional information. For LinLSTM, filtering sequences with only valid patterns of syllable weights during its beam decoding did not lead to any improvements. We find that larger beam sizes with a size of 512, as against 128 for the morphosyntactic task, lead to better results for this task. This difference can probably be attributed to larger path lengths of the final solution in a prosody task as compared to a morphosyntactic task due to the graph construction (Figure 8, merged Nodes). While the EBM configurations were trained on a data set of 10,000 verses from *Rāmāyaṇa*, the neural sequence generation models were trained on a data set of 110,000 verses. The additional 100,000 verses were taken from DCS and also from Vedabase.

Joint and Pipeline Modeling of Tasks. Tables 13–15 show the results for the three joint tasks we perform using the framework. For all the three tasks, we observe similar trends to what is observed in the previous research (Seeker and Çetinoğlu 2015; More et al. 2019), that the joint modeling of the tasks outperforms the pipeline-based models. In Table 13 we report the task of joint word segmentation and morphological parsing as reported in Krishna et al. (2018). The pipeline models perform far inferiorly than

Table 12
Results for dependency parsing.

System	UAS	LAS
EasyFirst	73.23	-
YAP	76.69	73.02
Tree-EBM-P	81.07	77.73
Tree-EBM-F	82.65	79.28
Tree-EBM*-F	**85.32**	**83.93**

Table 13
Results for the joint task of word segmentation and morphological parsing.

System	P	R	F	PM
EG-CRF	76.69	78.74	77.7	31.82
VL-EBM	76.88	74.76	75.8	27.49
BL-EBM	79.41	77.98	78.69	31.57
Prim-EBM-P	82.35	79.74	81.02	32.88
Pipe-Cliq-EBM-P	87.43	84.1	85.73	42.75
Joint-Cliq-EBM-P	91.35	89.57	90.45	55.78
Pipe-Cliq-EBM-F	87.23	85.38	86.29	44.81
Joint-Cliq-EBM-F	**94.04**	**91.47**	**92.74**	**58.21**

the corresponding joint models for the task. For Cliq-EBM-F the joint version reports a percentage improvement of 7.47% over its pipeline-based counterpart. The pipeline setting for the task first makes use of the Cliq-EBM-F word segmentation model to obtain the word splits, and the predicted words are then passed to SHR to generate the search space for morphological parsing. We also experimented with a pipeline setting where word segmentation was performed using rcNN-SS. The setting that used rcNN-SS for word-segmentation performed inferiorly to the pipeline setting that uses Cliq-EBM-F word segmentation model in terms of Precision, Recall, and F-Score. However, the former reports a better perfect match score of 45% against the 44.81% perfect match score of the latter. The perfect match score for the joint WS+MP task is less than that of the standalone word segmentation task by more than 25 percentage points. We observe that this is primarily due to syncretism and homonymy, as the system ended up predicting a different morphological tag or stem even though it predicted the correct inflected-form in those cases. Table 9 shows the results for morphological parsing when using gold standard segmented sentence as input.

Similarly, the results for the joint task of word segmentation, morphological parsing, and dependency parsing are shown in Table 14. It is evident that the pipeline system performs inferiorly to the joint modeling. The joint model of Prim-EBM*-F reports an F-Score of 79.2 as against an F-Score of 77.49 for the pipeline model. Table 15 shows the performance of pipeline and joint models for the joint task of word segmentation, morphological parsing, and syntactic linearization. Here, the pipeline-based model of Prim-EBM*-F is outperformed by the joint model with a significant margin of more than 11 BLEU points. In the pipeline configurations of both these tasks, we perform the word segmentation and morphological parsing jointly using Cliq-EBM-F/P models. For both the tasks, use of rcNN-SS for segmentation in the pipeline resulted in inferior performances, as compared to joint modeling of the upstream tasks using EBM.

4.3.1 Training Specifics.

Effect of Feature Function Learning Approach. FSPG, the feature function learning approach we introduce in this task, consistently outperforms PRA, previously used in Krishna et al. (2018), for all the tasks we experimented with (Section 4.3). Table 16 reports the Precision, Recall, and F-Score on the word segmentation task, for all the different feature

Table 14
Results for the joint task of word segmentation, morphological parsing, and dependency parsing.

System	P	R	F
Pipe-Prim-EBM-P	69.82	71.95	70.87
Joint-Prim-EBM-P	72.98	74.42	73.69
Pipe-Prim-EBM-F	75.17	76.58	75.87
Joint-Prim-EBM-F	77.65	77.92	77.78
Pipe-Prim-EBM*-F	76.54	77.39	76.96
Joint-Prim-EBM*-F	**78.68**	**79.72**	**79.2**

Table 15
Results for the joint task of word segmentation, morphological parsing, and syntactic linearization.

System	LCS	BLEU	PM (%)
Pipe-ATSP-EBM-F	38.51	32.78	10.14
Pipe-Beam-EBM-F	40.22	33.73	10.37
Pipe-Beam-EBM*-F	42.39	36.12	12.27
Joint-ATSP-EBM-P	46.75	36.67	11.83
Joint-Beam-EBM-P	51.44	40.18	13.92
Joint-ATSP-EBM-F	49.72	38.04	12.53
Joint-Beam-EBM-F	53.82	43.39	16.87
Joint-Beam-EBM*-F	**55.06**	**47.27**	**18.66**

sets we learn using FSPG and PRA. We find that the best performing set is the one using the FSPG and reports an F-Score of 98.35. Among the eight different configurations for PRA-based vectors (§4.2.5), the best configuration (Table 16, Row 3), by design, considers a feature space where all the features have $|\psi| \leq 1$. This implies all the features learned will have a constraint tuple of length at most 1. We observe a performance drop when we change this setting to $|\psi| \leq 2$ (Table 16, Row 4). On the contrary, FSPG does not require any such upper bound on the length of the constraint tuple ψ. This results in a quite diverse set of features. FSPG has only 54.94% of its features with a tuple size $|\psi| \leq 1$ for the morphosyntactic tasks. Interestingly, 17.06% of paths have $|\psi| \geq 3$, and the longest path was of size 4. In the case of prosody-level FSPG features, we find that all the features have a tuple size $|\psi|$ between 3 and 7. In fact more than 50% of the paths have a tuple size $|\psi| \geq 5$. Further, we learn feature sets of different sizes using both FSPG and PRA. For FSPG, we experimented with feature sets of sizes starting from 300 through 1,000, in step sizes of 50. We find that the model performs the best with a feature set size of 850 for morphosyntactic tasks, and 500 for the prosody-level task. Similarly, for PRA, we vary the feature set size from 400 to 2,000 in steps of 100, and

Table 16
Effect of feature selection approach tested on the word segmentation model.

Upper bound on $\|\psi\|$	Feature Selection	Label	P	R	F
1	MIR	PMI	92.86	95.27	94.05
2	MIR	PMI	82.18	86.17	84.13
1	MIR	Co-Occ	**94.48**	**96.52**	**95.49**
2	MIR	Co-Occ	93.65	95.52	94.57
1	RFE	PMI	80.93	87.65	84.16
2	RFE	PMI	84.85	86.90	85.86
1	RFE	Co-Occ	83.69	89.11	86.31
2	RFE	Co-Occ	86.34	89.47	87.88
-	FSPG	Co-Occ	**98.04**	**98.67**	**98.35**

Table 17
Performance of Cliq-EBM-F with increased number of candidates per inflected form in the input. First row ($k = 0$) shows the MP results on the original search space.

k	Similar tags	Disjoint tags
	F-score	
0	95.33	
2	92.12	93.71
3	91.03	93.08
5	88.89	91.14

Table 18
Performance of Cliq-EBM with pruned edges in G.

k	WS			WS+MP		
	P	R	F	P	R	F
5	90.46	92.27	91.36	83.52	80.48	81.97
10	92.92	95.07	93.98	85.32	84.4	84.86
15	94.85	96.14	95.49	87.67	86.38	87.02
20	**95.23**	**96.49**	**95.86**	**89.25**	**88.62**	**88.93**

we achieve the best results for a feature set with a size of 1,500. The feature function learning process takes about 9 to 11 hours when using the PRA approach (Krishna et al. 2018, supplementary). Using FSPG, the time required for feature function learning was reduced to about 4 hours for morphosyntactic tasks, and 3 hours for the prosody-level tasks. This is primarily due to the use of GreedyTree structure in FSPG, which avoids the need for exhaustive enumeration of all the features that was a bottleneck in PRA. While feature function learning is a one-time process that happens prior to training, the feature value computation is performed each time during the inference. Feature value computation is performed using PCRW, irrespective of the function learning approach used. However, owing to the lower number of features in the feature set when using FSPG, a feature vector for morphosyntactic tasks is generated in 38 ms as compared to 66 ms for PRA. For prosody-level tasks the time is about 23 ms (with 500 vector components).[32]

Overgeneration of Candidates in the Input Graph X. SHR is used for the generation of the search space X for the morphosyntactic tasks. For standalone morphological parsing, we use SHR for performing only the morphological analysis for the words in the input sequence, and not for the segmentation analysis. To understand the impact of the quality of the search space generated by SHR on our EBM models, we experiment with a scenario where we over-generate the possible analyzes each word can have in the search space. For the morphological parsing task, we add additional candidates to each word such that the added candidate would differ from existing candidates (from SHR) only in their morphological tags. Further, the morphological tags introduced for the additional candidates would be the same as the lexical category of the existing candidates.[33] Table 17 shows results of the Cliq-EBM-F morphological parsing model in two separate settings, namely, similar tags and disjoint tags. In similar tags, we add k additional analyzes for each word where the added analysis would differ from an existing candidate only by one grammatical category value. If an inflected form has the

32 The vector generation can be performed in parallel, and we use 40 threads in our experiments.
33 The inflected-form *Rāmaḥ, as* per SHR, can be analyzed either as a nominal or a verb. So a new added candidate for the inflected form would either be a nominal or a verb.

tag "nominative |1| masculine" in its analysis, then a newly added candidate would be "accusative |1| masculine." The latter, similar to the former, is a nominal and the latter differs from the former only in terms of its grammatical category value "case." In disjoint tags settings, the added nodes will have no grammatical category values shared with any of the candidate analyzes. If "nominative |1| masculine" is the only analysis for an inflected form, then a tag like "accusative |2| neuter" would be an eligible tag for the disjoint tags setting. Irrespective of the setting, the tags added are all valid morphological tags in Sanskrit and of the same lexical category as the candidates. We show instances with added 2, 3, and 5 analyzes per word. The morphological tagging performance degrades as the number of candidates increase in the search space. More interestingly, the system has greater difficulty in predicting the correct candidates when the candidates has similar tags as compared to having additional candidates with disjoint tags. We add k additional analyzes per inflected form in the input, implying the number of possible candidates in the search space increase by $n \times k$, where n is the number of words in the input.

Limiting the Context by Pruning Edges in the Input Graph X. From the tasks of word segmentation, morphological parsing, and their joint task formulation, we make two important observations. In these tasks, the EBM configurations that use inference procedures with non-sequential processing of input outperform those that perform the input sequentially. Between Prim-EBM and Cliq-EBM configurations, the two configurations with non-sequential processing, Cliq-EBM (which considers pairwise potentials between all the pairs of non-conflicting words) leads to significant performance improvements over Prim-EBM. But, to get a better understanding of the impact of pairwise potentials between non-conflicting words, we perform experiments where the edges are pruned from the original input graph X, if two words are separated by more than k characters between them.

For any two words appearing in an exhaustive segmentation, we keep an edge only if both the words overlap within a distance of k characters. We experiment with $k = 5$, 10, 15, and 20. Hence, for $k = 20$, a word will form edges with all the words that fall within 20 characters to the left and 20 characters to the right. The average length of an input sequence in the test data is 40.88 characters. We do not modify our inference procedure in Cliq-EBM-P other than to take care of the possibility that a clique need not always be returned. Table 18 shows the results for different values of k. Interestingly, the results show a monotonic increase with the increase in the context window size, and the results with the entire context are still better than those with $k = 20$, even though only marginally. It is interesting to note that keeping the entire context does not adversely affect the predictions as none of the pruned models outperform the original configuration in Cliq-EBM-P. The lattice structure can be seen as an extreme case of pruning. Similarly, we perform WS using Prim-EBM-P, where the model uses a non-sequential inference. However, we experiment with a new setting, where we use the same observed variable X constructed for VL-EBM (i.e., edges exist only between adjacent nodes), instead of the dense graph structure that is normally used for Prim-EBM-P. The lattice models used special start and end marker nodes, owing to their sequential inference. These nodes were removed from Prim-EBM-P. We find that the Prim-EBM-P with modified observed variable from VL-EBM reports an F-Score of 87.4 for WS. This implies Prim-EBM-P, when using the observed variable of VL-EBM, outperforms the VL-EBM configuration by more than two points. We find that VL-EBM makes more early mistakes, owing to its sequential inference, resulting in higher error propagation as compared to Prim-EBM-P.

Out of Vocabulary Stems and Inflected Forms in $\mathcal{C}$. For joint word segmentation and morphological parsing we use DCS10k as the test data. DCS10k has 8,007 stems, of which 514 (6.42%) are out of vocabulary in $\mathcal{C}$. For such stems in $\mathcal{C}$, we apply add-one smoothing prior to computation of the co-occurrence probability using PCRW. The model reports a macro-averaged F-Score of 92.74. The micro-averaged F-Score for those 514 OOV stems is 57.98. However, the F-Score quickly picks up even for those stems whose inflections occur only at most five times. We find that 833 in DCS10k occur at most five times in $\mathcal{C}$, and the micro-averaged F-Score for those is 72.87. The test set for dependency parsing (DP) has 100% stem coverage with vocabulary of $\mathcal{C}$. We create two synthetic versions of $\mathcal{C}$, where 5% and 10% of stems in the vocabulary of the DP test data are removed from $\mathcal{C}$. Tree-EBM*-F reports a UAS (LAS) of 84.95 (83.51) when using $\mathcal{C}$ with 5% missing stems. This setting stands closer to DCS10k in terms of missing stems. Similarly, Tree-EBM*-F reports a UAS (LAS) of 84.37 (83.04) when using $\mathcal{C}$ with 10% missing stems. Tree-EBM*-F originally reported a UAS (LAS) of 85.32 (83.93), when using $\mathcal{C}$ with 100% stem coverage. For a morphologically rich language like Sanskrit, the number of missing inflected forms for a stem that is present in the corpus graph vocabulary would be much higher than missing stems in the vocabulary. The DP test data has 13.81% of the inflected forms missing from $\mathcal{C}$. We create a synthetic version of $\mathcal{C}$, where we increase the missing inflected forms by 10% (i.e., to 23.81%). However, the performance drop is negligible as it drops from 85.32 to 85.09 UAS and from 83.93 to 83.77 LAS. For this experiment, we used $\mathcal{C}$ with 100% coverage.

Morphological Class Specific Assessment. Table 19 presents the micro-averaged recall for the words grouped based on their inflectional classes (Table 3) for Clique-EBM-F, Clique-EBM-P, and Prim-EBM-P. For word segmentation (WS), the surface-forms belonging to indeclinables and nominals have the highest recall. In the case of joint word segmentation and morphological parsing (WS+MP), the recall over nominals (surface-form and tag) shows the highest decline among the lexical cateogries. This difference in recall for nominals mostly arise due to mispredictions in the morphological tag of the word, rather than the surface-form (i.e., due to syncretism). The entry "compound" in Table 19 shows the recall for the non-final components of a compound (i.e., on those components where no inflection is applied). This entry was added due to the extensive use of compounds, often with more then two components, in Sanskrit. We find that considering the pairwise potential between all the words in a sentence in Clique-EBM-P/F led to improved morphological agreement between the words in comparison

Table 19
System performance in terms of Recall on the lexical categories (Table 3) for the competing systems Clique-EBM-P, Cliq-EBM-F, and Prim-EBM-P.

Type	WS Recall			WS + MP Recall		
	Prim-EBM -P	Cliq-EBM -P	Cliq-EBM -F	Prim-EBM -P	Cliq-EBM -P	Cliq-EBM -F
Nominal	93.06	96.87	98.07	86.14	89.0	91.33
Verb	89.14	95.91	96.84	87.38	94.42	95.14
Participle	87.66	96.42	97.17	87.12	94.71	95.52
Compound	89.35	93.52	95.69	86.01	91.07	92.86
Indeclinable	95.07	97.09	98.26	94.93	96.47	97.95

Table 20
Error propagation, in terms of the number of errors, in the pipeline (pipe) and joint models for word segmentation (WS), morphological parsing (MP), and dependency parsing.

Prim-EBM-F	Pipe	Joint	
		EBM	EBM*
WS	425	322	297
MP	703	547	520
Label mismatch	404	372	294
Others	886	827	777
Total errors	2,418	2,068	1,888

Table 21
Error propagation, in terms of the number of errors, in the pipeline (pipe) and joint models for word segmentation (WS), morphological parsing (MP), and syntactic linearization. Here dep. locality denotes dependency locality.

Beam-EBM-F	Pipe	Joint	
		EBM	EBM*
WS	318	233	204
MP	356	241	226
Dep. locality	213	197	144
Others	486	408	397
Total errors	1,373	1,079	971

to Prim-EBM-P. The difference is particularly visible in the case of improvements in the verb prediction. In Prim-EBM-P, the top five cases of mispredictions from one morphological class to a particular wrong class were due to syncretism. In Clique-EBM-P/F such patterns were no longer present and, more importantly, the skewedness in such mispredictions was considerably reduced.[34]

Error Propagation in the Pipeline and Joint Models for Word Segmentation, Morphological Parsing, and Dependency Parsing. Table 20 shows the error analysis over the entire test data of dependency parsing. We analyze the pipeline model and both the joint models, Prim-EBM-F and Prim-EBM*-F, for the task. As we can expect, the largest gap in errors between the pipeline model and the joint model is due to the mispredictions in word segmentation and in morphological parsing. At the same time, the reduction in mispredictions in the remaining categories of errors between these models is lower. Now, among the joint models, Prim-EBM*-F (marked as EBM*) shows considerable reduction in errors under the "label mismatch" category as compared to Prim-EBM-F. Prim-EBM*-F is the EBM configuration with language-specific augmentations as described in Section 3.4. Here, the "label mismatch" category specifically looks into those mispredictions that occur due to the many-to-many mapping between the case markers of the words and the dependency tags (*kāraka*). In this category, we consider those mispredictions where the gold tag and the mispredicted tag are applicable for the case markers of the words and hence are more susceptible to being mispredicted. A nominal, which is in instrumental case, can be assigned a *kartā* or a *karaṇam* relation (and rarely a *hetuḥ*). Similarly, among other cases, a nominal in nominative can also be assigned a *kartā* relation. However, two nominals, one in instrumental and another in nominative, both together cannot be a *kartā* in a sentence. Of the 204 occurrences of *karaṇam* in the test data, 63 of them were wrongly assigned as *kartā*. All these 63 cases had more than one prediction of *kartā* in the sentences. With Tree-EBM*-F, 56 of the cases were correctly assigned as *karaṇam*, and 7 were wrongly assigned as *hetuḥ*. Such cases have

34 Please refer to Tables 6 and 7 in the supplementary material of Krishna et al. (2018) for details.

substantially led to reduction in label mismatch for Tree-EBM*-F. This tag forms the majority contributor to the reduction due to label mismatch.

Error Propagation in the Pipeline and Joint Models for Word Segmentation, Morphological Parsing, and Syntactic Linearization. Table 21 shows the performance difference between the pipeline-based model and the joint models for the poetry to prose linearization task. Here, we perform a qualitative analysis on a sample of 300 sentences of the test data for linearization.[35] It can be observed that the pipeline model suffers due to errors from the word segmentation and morphological parsing. Among the joint models, Beam-EBM*-F makes considerably fewer errors in the dependency locality category as compared to the errors made by Beam-EBM-F. This is due to the language-specific augmentations (Section 3.4) made to Beam-EBM*-F that mitigate the errors from dependency locality. We observe that 39.62% of the reduction in error in this category is achieved specifically in the case of placement of adjectives near the word it modifies. The error reduction is primarily a result of pruning out candidates with longer dependency lengths and pruning of candidates with more than two violations of weak non-projectivity.

Merging of Syllable-Level Nodes in Prosodification. For the graph generation in the prosody-level tasks (Section 3.1), we define the vertices at a syllable level. At the same time, we have made a modification by merging the nodes that have only one outgoing edge (Figure 8). This node merging substantially reduces the number of nodes and edges in the input graph X, as well as the edge vectors generated. Empirically we find that this modification results in an overall improvement of system performance. While the best performing model of ours using the modified structure performs with a BLEU score of 46.88 and a Kendall's Tau score of 68.4, the original syllable-level representation results in a BLEU of only 41.93 and a Kendall's Tau score of 64.21.

Summarily, FSPG significantly and consistently outperforms PRA in learning effective feature functions for all the tasks. Our non-sequential method of inference results in better performance in comparison to the sequential models, even for low-level tasks such as word segmentation and morphological parsing. For morphological parsing, leveraging the pairwise potentials between every connected nodes while making a prediction results in reduced syncretism. The performance gain of Clique-EBM over Prim-EBM illustrates the effectiveness of this approach. Joint modeling of the tasks mitigates the error propagation from upstream tasks, compared to a pipeline-based approach. Language-specific augmentations in pruning the search space and filtering the candidates during inference reduces errors due to label mismatch and dependency locality for dependency parsing and syntactic linearization, respectively.

4.4 Experiments on Czech

Our proposed structured prediction framework is language-agnostic. We show its effectiveness by training models for morphological parsing and dependency parsing in Czech, another MRL.

35 We restricted ourselves to a sample of the test data due to the requirement of manual inspection of data to detect the errors in the "dependency locality" (Gibson et al. 2019) category. One of the authors and two other annotators with a graduate level degree in Sanskrit linguistics performed this. We filter 300 ($\times$ 3, as there are predictions from three systems to be considered) sentences where at least two of the three annotators agreed on the errors made by the system.

Tasks. We train models in Czech for three tasks, one related to morphological parsing and two related to dependency parsing. For morphological parsing, we follow the experimental and evaluation settings of the "Morphological Analysis and Lemmatization in Context" task conducted as part of the SIGMORPHON 2019 shared task (McCarthy et al. 2019). The task expects a system to input a sentence and output the lemma and morphological description, that is, the morphological tag, for each word in the sentence. This task is exactly the same as the morphological parsing task we perform for Sanskrit, and henceforth we refer to this task as morphological parsing. Similarly, for dependency parsing, we follow the experimental and evaluation settings of the "CoNLL 2018 Shared Task on Parsing Raw Text to Universal Dependencies" (Zeman et al. 2018), henceforth to be referred to as Raw2UD. Here, the participants were asked to parse sentences in a raw text document where no gold-standard pre-processing information (tokenization, lemmas, morphology) is available. The participating systems should output the labeled syntactic dependencies between the words (Zeman et al. 2018). To enable the participants to focus only on selected parts of the pipeline, predictions from a baseline system was provided for each prepossessing step (Straka and Straková 2017). We make use of the tokenization output from Straka and Straková (2017) and feed it to a morphological analyzer, whose output is then provided as input to our joint morphological and dependency parsing model, Prim-EBM-F. Finally, we also perform the standalone dependency parsing task, where sentences with gold tokenization and morphological description are provided as input.

Data. Both the shared tasks use treebanks from Universal Dependencies (UD; Nivre et al. 2018). The morphological parsing task (McCarthy et al. 2019) used the UD version 2.3[36] and Raw2UD (Zeman et al. 2018) used the UD version 2.2[37]. We perform our experiments on Czech, specifically on the Czech-PDT treebank available as part of UD. However, the train, dev, and test splits provided for both these shared tasks differ. We use the corresponding splits used in the shared tasks for reporting our model performances, such that our models are comparable with the models submitted as part of the shared task. For the dependency parsing (with gold tokenization and morphology), we use the train, test, and dev split of Czech-PDT in UD version 2.2. A morphologically tagged corpus is required for the construction of the corpus graph $\mathcal{C}_{Czech}$, which is used for generating the edge vectors. First, we construct $\mathcal{C}_{Czech}$ by only using the sentences from the PDT training data split.[38] We expand $\mathcal{C}_{Czech}$ by using one million raw sentences randomly sampled from the common crawl corpus. The relevant tokenization and morphological information, for these sentences, required to construct $\mathcal{C}_{Czech}$ was predicted using UDPipe (Zeman et al. 2018). The organizers of Raw2UD had already provided this data as part of the shared task.

Evaluation Metrics. McCarthy et al. (2019) report the performance of the participating systems based on lemma accuracy, lemma edit distance, tag accuracy, and tag F1-score. Here, lemma accuracy and tag accuracy report a 0/1 accuracy of the predicted lemma and morphological information, respectively, micro-averaged at the corpus level

36 https://github.com/sigmorphon/2019.

37 https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2837.

38 For each model, we used the corresponding training data split from the shared-task. For Raw2Ud the training set has 68.496 sentences while MP has 70,331 sentences in the training set.

(McCarthy et al. 2019).[39] The lemma edit-distance and tag F1-score were used to give partial credits to the systems, if they predict a part of the solution correctly. Because our system's predictions are basically selections of candidates from a morphological analyzer and not exactly character-wise or grammatical category–wise prediction, we use only the tag and lemma accuracy as evaluation metrics. For Raw2UD we use the official evaluation script used in the shared task. From the evaluation, we report the UAS and LAS scores. However, these UAS and LAS scores take the possible misalignment of the tokens in the prediction and in the ground truth, due to tokenization errors, also into consideration. For such misaligned tokens, the prediction is considered to be invalid. To avoid confusions with our previously defined UAS/LAS metric we would mark them with UAS^{tok} and LAS^{tok}. For standalone dependency parsing, we use UAS and LAS as evaluation metrics.

Models. For the morphological parsing task, we use the Cliq-EBM-F model, that is, the one that uses the maximal clique search as inference and FSPG for the edge vector generation. We use two variants, Cliq-EBM-F-Xtra and Cliq-EBM-F-PDT, where the former uses one million sentences from common-crawl and training data from PDT treebank for the corpus graph construction, and the latter uses only the PDT training data for the corpus graph construction. We consider three systems from McCarthy et al. (2019) as baselines for the task. The systems are CHARLES-SAARLAND-02 (Kondratyuk 2019), NLPCUBE-01 (Boros, Dumitrescu, and Burtica 2018), and EDINBURGH-02 (McCarthy et al. 2019).[40] Here, CHARLES-SAARLAND-02, henceforth to be referred to as SAARLAND-02, uses two separate word-level bidirectional residual LSTM decoders for the lemma and morphological tag prediction tasks. The model uses a common encoder and a separate attention layer for both the tasks. The encoder representations use multilingual BERT embeddings fine-tuned on a concatenation of all the available languages. This model achieved the best overall results for the task (in terms of Morph accuracy and F1-score) and also in the Czech-PDT treebank. Similarly NLPCUBE-01 and Edinburgh-02 achieved the third best scores for Czech-PDT treebank in terms of lemma accuracy and tag accuracy, respectively.[41] Edinburgh-02 is a character-level LSTM encoder-decoder model, while NLPCUBE-01 is a pipeline model that uses an encoder-decoder model for lemmatization and a LSTM-based sequence tagger for morphological tagging. Both these models use the same set of input features, which is a concatenation of character-level embeddings, word embeddings, and fasttext embeddings. These two models do not use any external resources and their models are trained using resources provided in the shared task.

For dependency parsing we train the Tree-EBM-F model, and for the Raw2UD task we use the Prim-EBM-F model. Tree-EBM-F uses the exact search Edmonds-Chu-Liu algorithm and Prim-EBM-F uses the approximate algorithm for maximal spanning tree as the inferences, respectively. The Raw2UD task is similar to the joint MP and DP task, where the search space considers all the possible morphological analyzes for a given word and hence the Prim-EBM-F model searches for a maximal spanning tree. Similar to morphological parsing task, we report scores for two variants of Prim-EBM-F for the Raw2UD task as well, Prim-EBM-F-PDT and Prim-EBM-F-Xtra. We consider the

39 https://sigmorphon.github.io/sharedtasks/2019/task2/.

40 NLPCUBE-01 and EDINBURGH-02 did not publish their findings as separate papers, and hence their results can be found in McCarthy et al. (2019).

41 We do not include, UFAL-Prague-01 (Straka, Straková, and Hajic 2019) the second best system as it also uses BERT based pretraining similar to SAARLAND-02.

three systems HIT-SCIR (Che et al. 2018), TurkuNLP (Kanerva et al. 2018), and Stanford (Qi et al. 2018). These systems achieved first, second, and third positions in Czech-PDT treebank based on the LAS^{tok} metric. All the three systems use the neural biaffine parser proposed by Dozat and Manning (2017) with minor modifications. Che et al. (2018) makes use of ELMO embeddings trained on 20 million words from the common-crawl Czech corpus. TurkuNLP and Stanford use pretrained word2vec embeddings trained on the common-crawl corpus (Ginter et al. 2017). All the models for Czech use MorphoDita (Straková, Straka, and Hajič 2014) as the morphological analyzer (i.e., the graph-generator). For the dependency parsing task we use Stanza (Qi et al. 2020), an extended version of Qi et al. (2018), and UDPipe (Straka and Straková 2017) as the baselines.

Results. Table 22 shows the result for the morphological parsing task. Our model Cliq-EBM-F-Xtra stands third in terms of lemma accuracy and tag accuracy in comparison to all the participating systems in the shared task (McCarthy et al. 2019). The first two systems in the shared task, SAARLAND-02 and UFAL-PRAGUE-01, greatly benefit from the fine-tuning and pretraining, respectively, on the multilingual contextualized BERT embeddings. Among the systems that use no external resources other than the training data, our model Cliq-EBM-F-PDT scores the highest in terms of tag accuracy, followed by NLPCUBE-01 and EDINGBURGH-02, respectively. Our morphological analyzer identifies 99.74% of the lemma in the test data for the shared task. Similarly, 99.45% of the morphological tags were identified by our morphological analyzer.

Table 23 shows the results for the Raw2UD task. Our system Prim-EBM-F-Xtra achieves the best result, outperforming all the models in the shared task. Similarly, for the dependency parsing task, Stanza (Qi et al. 2020) reports 93.73 UAS and 92.19 LAS on the Czech-PDT treebank. UDPipe reports 90.3 UAS and 88.2 LAS. Our proposed model Tree-EBM-F-Xtra reports 95.84 UAS and 93.67 LAS. Tree-EBM-F-PDT reports a UAS of 94.38 and LAS of 91.28. Here, both Tree-EBM-F-Xtra report the best score, outperforming all the baselines. Tree-EBM-F-PDT, the model with no external resources

Table 22
Results for the SIGMORPHON 2019 morphological analysis in context shared task. All the non-EBM systems were competing systems in the shared task (McCarthy et al. 2019). SAARLAND-02 corresponds to the CHARLES-SAARLAND-02 in the shared task and other systems follow the same naming as in the shared task.

System	Lemma Accuracy	Morph Accuracy
SAARLAND-02	**99.42**	**98.54**
EDINBURGH-02	98.66	93.90
NLPCUBE-01	98.50	95.89
Cliq-EBM-F-PDT	98.16	95.97
Cliq-EBM-F-Xtra	98.94	97.21

Table 23
Results for the Raw2UD task. All the Non-EBM systems were the competing systems in the shared task (Zeman et al. 2018). The evaluation metrics UAS^{tok} and LAS^{tok} considers the misalignment due to segmentation and tokenization as well.

System	UAS^{tok}	LAS^{tok}
HIT-SCIR	93.44	91.68
TurkuNLP	92.57	90.57
Stanford	92.44	90.38
Prim-EBM-F-PDT	92.67	90.88
Prim-EBM-F-Xtra	**93.76**	**92.03**

other than the training data, outperforms both the neural baseline models in terms of UAS. Overall, we find that our EBM models achieve competitive results in all the three tasks, outperforming all the baselines in two of the three tasks.

5. Discussion

Our EBM framework consists of 3 components, namely, a graph generator, an edge vector generator, and a structured prediction model. All the three components act as a pipeline, where the edge vector generator obtains its input from graph generator, and the structured prediction model obtains its inputs from both the other components. This separation of the tasks becomes beneficial at multiple steps. First, we have limited data for both the syntax level tasks, dependency parsing and syntactic linearization, in Sanskrit. As demonstrated, the feature function we use substantially reduces the task-specific training data requirements. Second, our edge vectors are task agnostic,[42] thereby enabling the use of the same feature vectors for standalone and joint formulation of multiple tasks. Finally, such a separation enables us to plug and play multiple third party external tools such as the graph generator to our framework. For instance, while SHR was the graph generator module for Sanskrit, MorphoDita was used for Czech, and both of these are external tools developed independent of our framework.

Our feature function learning approach using FSPG can be seen as a pretraining approach, similar to the learning of word embeddings or network embeddings. In information networks terminology, the corpus graph we construct is an instance of a Heterogeneous Information network (Sun 2010), where our corpus graph contains nodes of different types. The typed paths we learn are metapaths and we use random walk-based inference to compute a scalar feature value. A major novelty of our work is the automated generation and filtering of the typed paths that compute the strength of association between two nodes under diverse constraints. A large number of neural embedding approaches are either intended for homogeneous information networks, or are used to learn embeddings for only one metapath at a time (Dong, Chawla, and Swami 2017). Recently, graph neural networks were introduced for learning network embeddings using metapaths. These approaches generally tend to use a limited number of metapaths, ranging from two to six (Cen et al. 2019; Fu et al. 2020). Most importantly, these approaches still assume the metapaths are hand-crafted, while we automate the identification of relevant metapaths.

An immediate future work would be the extension of the proposed framework to perform downstream semantic tasks such as semantic role labeling. The tagging scheme that we used for the dependency analysis is based on the *kāraka* theory, and the relations as per *kāraka* theory are known to be syntactic-semantic in nature (Bharati and Sangal 1993). This was a conscious design decision we made, which leaves us with a potential scope for expanding the framework for a semantic labeling task. Given the link between the case markers to *kāraka* tags as per traditional Sanskrit grammar, we hypothesize that modeling of semantic labeling tasks such as Semantic role labeling might benefit from jointly modeling it with its preliminary tasks. However, task-specific labeled data for such tasks are scarce or non-existent in Sanskrit. Even for syntax level tasks, such as DP and SL, we observed that we have limited availability for task-specific labeled data and this gets worse for tasks further down the processing pipeline for Sanskrit.

42 Our vectors need not necessarily be domain-agnostic as we use different edge vectors for morphosyntactic and prosodic tasks.

For all the tasks we experimented with in the framework, the models used a relatively limited amount of task-specific labeled data for Sanskrit, in comparison to other neural baselines. However, this is enabled due to the availability of a morphologically tagged corpus, DCS, and the use of SHR, a lexicon driven shallow parser. SHR is beneficial to our framework in multiple ways. One, its efficient implementation coupled with a wide coverage lexicon enables us to exhaustively enumerate various possible word splits for a given sequences. Two, the morphological analysis provided by SHR can be quite detailed. This may involve additional information about derivational nouns, currently restricted to nominals derived from verbs. A nominal derived from a verb, when derived from specific derivational affixes, may act as a verbal head to other words in Sanskrit.[43] Three, it can also encode additional morphosyntactic information, in addition to the grammatical categories for several words, particularly verbs.[44] Four, it may generate only those inflected forms where the morphemes involved in the creation of the form are compatible to each other as per rules of Sanskrit, thereby avoiding overgeneration of candidates. To elaborate, preverbs are applied to a root or the stem of a derivational noun (derived from verb), only after checking their mutual compatibility. Similarly, a form *ambaraḥ* will be invalid as per SHR because the stem *ambara* can form inflections only for forms in neuter gender. However the compound *pītāmbaraḥ* with component stems *pīta* and *ambara* is allowed.[45] This is because *pītāmbaraḥ* is an exocentric compound (*bahuvrīhi*), and an inflection in masculine gender is possible for the compound.

Our EBM framework is language-agnostic in general as evidenced from our experiments in Sanskrit and Czech. Using our framework, we can train models for another language, say for morphosyntactic tasks, provided we have access to a morphological dictionary or a morphological analyzer for the language. Further, we also need access to a morphologically tagged corpus to build our corpus graph for feature function learning. It needs to be noted that a shallow parser like SHR might not be readily available for several languages. However, this necessarily need not limit the functionality of our framework. We successfully trained highly competitive models in Czech using the MorphoDita morphological dictionary (Straková, Straka, and Hajič 2014), which is limited in its functionality as compared to SHR. Further, we successfully constructed the corpus graph for Czech using 1 million raw sentences with predicted morphological information, along with 70,000 gold standard morphologically tagged sentences. For Sanskrit, we used 350,000 morphologically tagged textlines from DCS. For the learning of feature functions, the only domain knowledge we require is the knowledge of grammatical categories and what combinations of grammatical categories can form a lexical category in the language. This information is used to define the types in the corpus graph, and the typed paths (features) are automatically learned using FSPG. Probably the most linguistically involved part of our framework is the use of constraints to prune the candidates in the search space that we use for Sanskrit (in models marked with a '*'). However, this is an optional component, and we show that highly competitive models can be trained, even without incorporating any such linguistic constraints, for both Sanskrit and Czech.

We assume that the framework can be potentially useful for processing texts in other free word order languages as well, as we demonstrated for Sanskrit and Czech. However, when extending the framework for a new language, one needs to keep in

43 We mention such a case in footnote 20.
44 We mention such a case in footnote 21.
45 SHR analysis available at `https://bit.ly/pitambara`.

mind that the framework uses non-sequential processing of input even for low-level tasks such as word segmentation and morphological parsing. Further, we construct dense graph structures as input. This makes the inference computationally costly. These design decisions were made considering the relatively free word order nature of sentences in Sanskrit, especially for verses. Owing to the computationally costly inference, our framework may not be well suited for languages that tend to follow a relatively fixed word ordering in their sentence constructions. Efficient and exact inferences that perform sequential processing of the input are available for such languages. More importantly, using our current input graph construction approach might result in ignoring valuable grammatical information encoded using word order in such languages.

6. Related Work

The last couple of decades have shown considerable interests in computational processing of Sanskrit texts, though mostly centered around formal language theory (Kiparsky 2009; Goyal and Huet 2016; Hyman 2009) and rule-based systems (Kulkarni 2013; Kulkarni et al. 2015; Goyal et al. 2012). Due to the low-resource nature of the language, attempts for developing data-driven models for processing Sanskrit texts has been less common, with a possible exception for the task of word segmentation. While early models for segmentation (Natarajan and Charniak 2011; Mittal 2010) focused on shorter sequences, typically with one or two split points, Krishna et al. (2016) proposed a segmentation model that deals with sentence-level inputs. Their model, similar to ours, use the search space from the Sanskrit Heritage Reader (Goyal and Huet 2016) and treats the problem as an iterative query expansion task. Hellwig (2015) was the first to pose the problem as a neural sequence labeling task, and later a considerably improved sequence labeler was presented in Hellwig and Nehrdich (2018). In Hellwig and Nehrdich (2018), the architecture incorporated recurrent and convolutional neural units, in addition to the use of shortcut connections. Both Reddy et al. (2018) and Aralikatte et al. (2018) proposed seq2seq models for the task. Aralikatte et al. (2018) considered the word segmentation as a multitask problem, using a common encoder with two decoders, where one decoder predicts the split location and the other is used to generate the characters in the split word.

Both word segmentation and morphological parsing are low-level, yet non-trivial tasks for multiple languages and are extensively researched in NLP. Traditionally, solutions for these tasks were proposed using (Probabilistic/Weighted) finite-state transducers (Kaplan and Kay 1981; Sproat et al. 1996; Mohri 1997; Beesley et al. 2003) and using unsupervised pattern discovery approaches (Goldsmith 2001; Argamon et al. 2004; Johnson and Goldwater 2009). Broadly, these approaches could be categorized as lexicon driven (Huet 2003; Chen and Liu 1992), purely statistical (Eisner 2002; Sproat et al. 1994), or both (Sproat et al. 1996). Sequence labeling approaches such as HMMs (Hakkani-Tur, Oflazer, and Tur 2000) and CRFs (Smith, Smith, and Tromble 2005; Xue 2003) were later proposed for these tasks. Further, lattice parsing-based approaches, where the input search space was represented as word-level lattices, were incorporated into CRFs (Smith, Smith, and Tromble 2005) for joint modeling of these tasks (Kudo, Yamamoto, and Matsumoto 2004). Neural sequence labeling approaches currently achieve state-of-the-art performance in word segmentation, especially for Chinese, Korean, Japanese, and so forth (Wang, Voigt, and Manning 2014; Shao, Hardmeier, and Nivre 2018). Similarly, higher-order CRFs (Müller, Schmid, and Schütze 2013) and neural morphological taggers (Malaviya, Gormley, and Neubig 2018; Tkachenko and Sirts 2018) are widely used for morphological parsing. Interestingly, lattice based

structures and lattice-parsing techniques remain hugely popular for word segmentation (Yang, Zhang, and Liang 2019) and in morphological parsing of morphologically rich languages (Seeker and Çetinoğlu 2015; More 2016; More et al. 2019).

Transition-based approaches (Kübler, McDonald, and Nivre 2009) and graph-based approaches (McDonald et al. 2005b) are primarily two approaches adopted for DP. The majority of the methods proposed for joint morphosyntactic parsing also fall into one of these categories. Such approaches have proven to be effective for a host of morphologically rich languages such as Hebrew, Turkish, Czech, Finnish, Arabic, and so on. (Cohen and Smith 2007; Goldberg and Tsarfaty 2008; Bohnet et al. 2013). Hatori et al. (2012) formulated the problem of joint word segmentation, POS tagging, and DP in Chinese using a transition-based framework. Seeker and Çetinoğlu (2015) performed the joint morphological segmentation and analysis along with DP for Hebrew and Turkish. The approach constructs a sentence-level graph with word-level morphological lattices and performs a dual decomposition wherein the predicted dependency tree and the morphological paths need to be arrived at an agreement for the final solution. Recently, More et al. (2019) proposed a transition-based framework (Zhang and Clark 2011), which encompasses a joint transition system, training objective, and inference for the morphological processing and syntactic parsing tasks. The system outperforms their previous standalone transition-based system for morphological analysis (More and Tsarfaty 2016), emphasizing their benefits of joint morphosyntactic parsing.

For dependency parsing, Kiperwasser and Goldberg (2016a) proposed replacing the traditional feature engineering approach with vector representations of tokens obtained from BiLSTM-based neural architectures. Several neural parsing models have extended this idea (Kuncoro et al. 2017; Dozat and Manning 2017) and report competitive results for languages such as English. Nevertheless these models have shown to be of limited effectiveness as compared to feature engineered models for morphologically rich languages (More et al. 2019). In our framework, we do not use any hand-crafted features. Instead, we automate the learning of the feature function and formulate it as learning of horn clauses (Lao and Cohen 2010; Gardner and Mitchell 2015). The task of obtaining the features for the edge vector is similar to obtaining a distributional composition between two words (Weir et al. 2016). Our work stands close to the attempts such as algebraic formulation for feature extraction by Srikumar (2017) or the Monte Carlo Tree Search-based feature selection approach by Gaudel and Sebag (2010). In Krishna et al. (2018) we used the Path Ranking Algorithm (Lao and Cohen 2010), a random walk based approach for learning horn clauses across a heterogeneous information network (HIN). The type of horn clauses mentioned in PRA is widely known as metapaths (Sun 2010) in HINs (Shi et al. 2017). Traditionally, metapaths, like feature engineering, were manually constructed. But, recent approaches such as PRA and FSPG (Meng et al. 2015) automate the generation of metapaths.

Word linearization has been used in various Natural Language Generation tasks for the past three decades. Linearization tasks are generally solved by incorporating syntactic information (He et al. 2009), semantic information (Puduppully, Zhang, and Shrivastava 2017), or by using language models (Hasler et al. 2017). Syntactic linearization can further be categorized into full tree linearizaton (Zhang and Clark 2015; He et al. 2009) or partial tree linearization (Zhang 2013) depending on the amount of syntactic information used in the method. In language model-based linearization approaches, purely distributional information is used for the task. Recently, such approaches have been shown to outperform syntax-based linearization approaches in English (Schmaltz, Rush, and Shieber 2016; Wiseman and Rush 2016). Similar to Wiseman and Rush (2016), Krishna et al. (2019) propose a seq2seq model for syntactic linearization in Sanskrit.

The sequence-level model, along with two of its accompanying pretraining steps, have shown to outperform other linearization models, which were originally proposed for English.

7. Conclusion

Our work presents a generic search-based structured prediction framework for numerous structured prediction tasks in Sanskrit. As shown by our additional experiments in Czech, in addition to the experiments in Sanskrit, our framework is language-agnostic. In Czech, we outperform all the participating systems in Zeman et al. (2018) and we report the third best score in McCarthy et al. (2019). In Sanskrit, we either achieve state-of-the-art results or ours is the only data-driven model for all the tasks we experiment with. In fact, we introduce the task of prosodification. The framework, though language-agnostic, still enables us to encode language-specific constraints, which helps to prune the input search space as well as to filter the candidates during inference. It also facilitates joint modeling of multiple tasks, which we plan to extend to further downstream semantic tasks as well. But more importantly, we achieve substantial reduction in task-specific training data requirements for all the tasks by incorporating rich linguistic information, both in the form of pruning the search space as well as in designing our feature function. This is particularly important for a low resource language like Sanskrit. Another novel aspect in this work is the automated learning of feature function. For one, this shifts the burden of designing effective features for each task from human domain experts and automates it. The only domain information required here is to define the literals required for the generation of horn clauses. This makes it adaptable to different scenarios, as witnessed in the case of morphosyntactic and prosodic tasks in our case. The use of FSPG-based feature generation not only improved the system performance but it also substantially reduced the time taken, as this does not require an exhaustive enumeration of the features.

Acknowledgments

We are grateful to Oliver Hellwig for providing the DCS Corpus, Amba Kulkarni for providing the STBC corpus, Anupama Ryali for providing the Śiśupālavadha corpus, and Gérard Huet for providing the Sanskrit Heritage Engine, to be installed at local systems. We extend our gratitude to Amba Kulkarni, Peter Scharf, Gérard Huet, and Rogers Mathew for their helpful comments and discussions regarding the work. We thank the anonymous reviewers for their constructive and helpful comments, which greatly improved the article.

References

Altun, Yasemin, Mark Johnson, and Thomas Hofmann. 2003. Investigating loss functions and optimization methods for discriminative learning of label sequences. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 145–152, Sapporo. DOI: https://doi.org/10.3115/1119355.1119374

Apte, Vaman Shivaram. 1965. *The Practical Sanskrit-English Dictionary: Containing Appendices on Sanskrit Prosody and Important Literary and Geographical Names of Ancient India*, Motilal Banarsidass Publications.

Aralikatte, Rahul, Neelamadhav Gantayat, Naveen Panwar, Anush Sankaran, and Senthil Mani. 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. DOI: https://doi.org/10.18653/v1/D18-1530

Argamon, Shlomo, Navot Akiva, Amihood Amir, and Oren Kapah. 2004. Efficient unsupervised recursive word segmentation using minimum description length. In *Proceedings of Coling 2004*, pages 1058–1064, Geneva. DOI: https://doi.org/10.3115/1220355.1220507

Beesley, K. R. and L. Karttunen. 2003. *Finite State Morphology*. Number v. 1 in CSLI studies in computational linguistics: Center for the Study of Language and Information. CSLI Publications.

Belanger, David. 2017. *Deep Energy-Based Models for Structured Prediction*. Ph.D. thesis, University of Massachusetts Amherst.

Belanger, David, Bishan Yang, and Andrew McCallum. 2017. End-to-end learning for structured prediction energy networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 429–439.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Bharati, Akshar and Rajeev Sangal. 1993. Parsing free word order languages in the Paninian framework. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 105–111, Columbus, OH. DOI: https://doi.org/10.3115/981574.981589.

Bhatta, Vinayak P. 1990. Theory of verbal cognition (Sābdabodha). *Bulletin of the Deccan College Research Institute*, 49:59–74.

Bishop, Christopher M. 1995. *Neural Networks for Pattern Recognition*, Oxford University Press, Inc.

Bohnet, Bernd. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing.

Bohnet, Bernd, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. In *Transactions of the Association for Computational Linguistics*, 1:415–428.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146. DOI: https://doi.org/10.1162/tacl_a_00051

Boros, Tiberiu, Stefan Daniel Dumitrescu, and Ruxandra Burtica. 2018. NLP-cube: End-to-end raw text processing with neural networks. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 171–179, Brussels. https://www.aclweb.org/anthology/K18-2017.

Bron, Coen and Joep Kerbosch. 1973. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577. DOI: https://doi.org/10.1145/362342.362367

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 957–961, Prague.

Cen, Yukuo, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation learning for attributed multiplex heterogeneous network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1358–1368, New York. DOI: https://doi.org/10.1145/3292500.3330964

Che, Wanxiang, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels. https://www.aclweb.org/anthology/K18-2005.

Chen, Keh Jiann and Shing-Huan Liu. 1992. Word identification for Mandarin Chinese sentences. In *Proceedings of the 14th Conference on Computational Linguistics-Volume 1*, pages 101–107. DOI: https://doi.org/10.3115/992066.992085

Chu, Yoeng Jin and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Cohen, Shay B. and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 208–217, Prague. http://www.aclweb.org/anthology/D/D07/D07-1022.

Collobert, Ronan, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Coulson, Michael. 1976. *Sanskrit: An Introduction to the Classical Language*, Teach yourself books, Hodder and Stoughton. https://books.google.co.uk/books?id=CASFbwAACAAJ.

Das, Monali. 2017. *Discourse Analysis of Sanskrit Texts: First Attempt Towards Computational Processing*. Ph.D. thesis, University of Hyderabad, Hyderabad. https://shodhganga.inflibnet.ac.in/bitstream/10603/219349/1/01_title.pdf

Dong, Yuxiao, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 135–144. DOI: https://doi.org/10.1145/3097983.3098036

Doppa, Janardhan Rao, Alan Fern, and Prasad Tadepalli. 2014. Hc-search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research*, 50(1):369–407. DOI: https://doi.org/10.1613/jair.4212

Dozat, Timothy and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations*, pages 1–8, Toulon. https://openreview.net/forum?id=Hk95PK9le.

Dror, Rotem, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392, Melbourne. DOI: https://doi.org/10.18653/v1/P18-1128

Edmonds, Jack. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240. DOI: https://doi.org/10.6028/jres.071B.032

Edunov, Sergey, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Classical structured prediction losses for sequence to sequence learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 355–364. DOI: https://doi.org/10.6028/jres.071B.032

Efron, Bradley, Trevor Hastie, Iain Johnstone, Robert Tibshirani, and others. 2004. Least angle regression. *The Annals of Statistics*, 32(2):407–499. DOI: https://doi.org/10.1214/009053604000000067

Eisner, Jason. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, PA. DOI: https://doi.org/10.3115/1073083.1073085

Fu, Xinyu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, WWW '20, pages 2331–2341, New York. DOI: https://doi.org/10.1145/3366423.3380297, PMID: 32303395, PMCID: PMC7156245

Gardner, Matt and Tom Mitchell. 2015. Efficient and expressive knowledge base completion using subgraph feature extraction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1488–1498, Lisbon.

Gardner, Matt, Partha Talukdar, and Tom Mitchell. 2015. Combining vector space embeddings with symbolic logical inference over open-domain text. In *2015 AAAI Spring Symposium Series*, pages 61–65.

Gaudel, Romaric and Michele Sebag. 2010. Feature selection as a one-player game. *International Conference on Machine Learning*, pages 359–366.

Gehring, Jonas, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning, In *Proceedings of Machine Learning Research*, pages 1243–1252, Sydney. http://proceedings.mlr.press/v70/gehring17a.html

Gibson, Edward. 1998. Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68(1):1–76. DOI: https://doi.org/10.1016/S0010-0277(98)00034-1

Gibson, Edward, Richard Futrell, Steven T. Piandadosi, Isabelle Dautriche, Kyle Mahowald, Leon Bergen, and Roger Levy. 2019. How efficiency shapes human language. *Trends in Cognitive Sciences*, 23(5):389–407. DOI: https://doi.org/10.1016/j.tics.2019.02.003, PMID: 31006626

Gibson, Edward, Steven T. Piantadosi, Kimberly Brink, Leon Bergen, Eunice Lim, and Rebecca Saxe. 2013. A noisy-channel account of crosslinguistic word-order variation. *Psychological Science*, 24(7):1079–1088.

Gildea, Daniel and David Temperley. 2010. Do grammars minimize dependency length? *Cognitive Science*, 34(2):286–310.

Gillon, Brendan and Benjamin Shaer. 2005. Classical Sanskrit, 'wild trees', and the

properties of free word order languages. In Katalin É. Kiss, editor, *Universal Grammar in the Reconstruction of Ancient Languages*. De Gruyter Mouton, pages 457–494.

Ginter, Filip, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. 2017. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Goldberg, Yoav and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, CA.

Goldberg, Yoav and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proceedings of ACL-08: HLT*, pages 371–379, Columbus, OH.

Goldsmith, John. 2001. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198.

Goyal, Pawan and Gerard Huet. 2016. Design and analysis of a lean interface for Sanskrit corpus annotation. *Journal of Language Modelling*, 4(2):145–182. DOI: https://doi.org/10.15398/jlm.v4i2.108

Goyal, Pawan, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for Sanskrit processing. In *Proceedings of COLING 2012*, pages 1011–1028.

Guyon, Isabelle, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1–3):389–422.

Hakkani-Tur, Diiek Z., Kemal Oflazer, and Gokhan Tur. 2000. Statistical morphological disambiguation for agglutinative languages. *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*, pages 285–291. DOI: https://doi.org/10.3115/990820.990862

Hasler, Eva, Felix Stahlberg, Marcus Tomalin, Adria de Gispert, and Bill Byrne. 2017. A comparison of neural models for word ordering. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 208–212. DOI: https://doi.org/10.18653/v1/W17-3531

Hatori, Jun, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 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. http://www.aclweb.org/anthology/P12-1110.

He, Wei, Haifeng Wang, Yuqing Guo, and Ting Liu. 2009. Dependency based Chinese sentence realization. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 809–816, Suntec. DOI: https://doi.org/10.3115/1690219.1690260

Heigold, Georg, Guenter Neumann, and Josef van Genabith. 2017. An extensive empirical evaluation of character-based morphological tagging for 14 languages. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 505–513. DOI: https://doi.org/10.18653/v1/E17-1048

Hellwig, Oliver. 2010–2016. *DCS - The Digital Corpus of Sanskrit*. Berlin. http://kjc-sv013.kjc.uni-heidelberg.de/dcs/

Hellwig, Oliver. 2015. Using recurrent neural networks for joint compound splitting and Sandhi resolution in Sanskrit. In *Proceedings of the 7th Language & Technology Conference*, pages 289–293.

Hellwig, Oliver and Sebastian Nehrdich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763. DOI: https://doi.org/10.18653/v1/D18-1295

Hirakawa, H. 2001. Semantic dependency analysis method for Japanese based on optimum tree search algorithm. In *Proceedings of the PACLING2001*, pages 117–126.

Hock, Hans Henrich. 2015. Some issues in Sanskrit syntax. In Peter Scharf, editor, *Sanskrit Syntax*, Sanskrit Library, Université Paris Diderot, pages 1–52.

Horvat, Matic and William Byrne. 2014. A graph-based approach to string regeneration. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for*

Computational Linguistics, pages 85–95, Gothenburg. DOI: https://doi.org/10.3115/v1/E14-3010

Huet, Gérard. 2003. Zen and the art of symbolic computing: Light and fast applicative algorithms for computational linguistics, In *Practical Aspects of Declarative Languages*, pages 17–18, Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/3-540-36388-2_3

Huet, Gérard. 2005. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *Journal of Functional Programming*, 15(4):573–614. DOI: https://doi.org/10.1017/S0956796804005416

Huet, Gérard and Amba Kulkarni. 2014. Sanskrit linguistics Web services. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 48–51, Dublin.

Hyman, Malcolm D. 2009. Sanskrit computational linguistics. *PāNinian Sandhi to Finite State Calculus*, Springer, pages 253–265, Dublin.

Ishikawa, Hiroshi. 2011. Transformation of general binary MRF minimization to the first-order case. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1234–1249. DOI: https://doi.org/10.1109/TPAMI.2010.91, PMID: 20421673

Johnson, Mark and Sharon Goldwater. 2009. Improving nonparameteric Bayesian inference: experiments on unsupervised word segmentation with adaptor grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 317–325, Boulder, CO. DOI: https://doi.org/10.3115/1620754.1620800

Kanerva, Jenna, Filip Ginter, Niko Miekka, Akseli Leino, and Tapio Salakoski. 2018. Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 133–142, Brussels.

Kaplan, Ronald M. and Martin Kay. 1981. Phonological rules and finite-state transducers. In *Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting*, pages 27–30.

Kiela, Douwe, Changhan Wang, and Kyunghyun Cho. 2018. Dynamic meta-embeddings for improved sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1477. DOI: https://doi.org/10.18653/v1/D18-1176

Kiparsky, P. and J. F. Staal. 1969. Syntactic and semantic relations in Pāini. *Foundations of Language*, 5(1):83–117.

Kiparsky, Paul. 1995. Pāinian linguistics. In *Concise History of the Language Sciences*, Elsevier, pages 59–65. DOI: https://doi.org/10.1016/B978-0-08-042580-1.50012-X

Kiparsky, Paul. 2009. On the architecture of Pāini's grammar. *Sanskrit Computational Linguistics*, pages 33–94, Springer, Berlin, Heidelberg.

Kiperwasser, Eliyahu and Yoav Goldberg. 2016a. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327. DOI: https://doi.org/10.1162/tacl_a_00101

Kiperwasser, Eliyahu and Yoav Goldberg. 2016b. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327. DOI: https://doi.org/10.1162.1162/tacl_a_00101

Kohavi, Ron and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324. DOI: https://doi.org/10.1016/S0004-3702(97)00043-X

Kondratyuk, Dan. 2019. Cross-lingual lemmatization and morphology tagging with two-stage multilingual BERT fine-tuning. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 12–18, Florence. DOI: https://doi.org/10.18653/v1/W19-4203

Kraskov, Alexander, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Physical Review E*, 69(6):69–85. DOI: https://doi.org/10.1103/PhysRevE.69.066138, PMID: 15244698

Krishna, Amrith, Bishal Santra, Sasi Prasanth Bandaru, Gaurav Sahu, Vishnu Dutt Sharma, Pavankumar Satuluri, and Pawan Goyal. 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. DOI: https://doi.org/10.18653/v1/10x-12768

Krishna, Amrith, Bishal Santra, Pavankumar Satuluri, Sasi Prasanth Bandaru, Bhumi Faldu, Yajuvendra Singh, and Pawan Goyal. 2016. Word segmentation in Sanskrit using path constrained random walks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 494–504.

Krishna, Amrith, Pavan Kumar Satuluri, and Pawan Goyal. 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. DOI: https://doi.org/10.18653/v1/W17-2214

Krishna, Amrith, Vishnu Sharma, Bishal Santra, Aishik Chakraborty, Pavankumar Satuluri, and Pawan Goyal. 2019. Poetry to prose conversion in Sanskrit as a linearisation task: A case for low-resource languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1160–1166, Florence. DOI: https://doi.org/10.18653/v1/P19-1111

Kübler, Sandra, Ryan McDonald, and Joakim Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127. DOI: https://doi.org/10.2200/S00169ED1V01Y200901HLT002

Kudo, Taku. 2006. Mecab: Yet another part-of-speech and morphological analyzer. http://mecab.sourceforge.jp

Kudo, Taku, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of EMNLP 2004*, pages 230–237, Barcelona.

Kulkarni, Amba. 2013. A deterministic dependency parser with dynamic programming for Sanskrit. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 157–166.

Kulkarni, Amba, Sheetal Pokar, and Devanand Shukl. 2010. Designing a constraint based parser for Sanskrit. In *International Sanskrit Computational Linguistics Symposium*, pages 70–90. DOI: https://doi.org/10.1007/978-3-642-17528-2_6

Kulkarni, Amba and K. V. Ramakrishnamacharyulu. 2013. Parsing Sanskrit texts: Some relation specific issues. In *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium*.

Kulkarni, Amba and Dipti Sharma. 2019. Pāinian syntactico-semantic relation labels. In *Proceedings of the Fifth International Conference on Dependency Linguistics (Depling, SyntaxFest 2019)*, pages 198–208, Paris. DOI: https://doi.org/10.18653/v1/W19-7724, PMCID: PMC6699228

Kulkarni, Amba, Preethi Shukla, Pavankumar Satuluri, and Devanand Shukl. 2015. How free is free word order in Sanskrit. *The Sanskrit Library, USA*, 269–304, https://goo.gl/7GnXuS

Kulkarni, Malhar, Chaitali Dangarikar, Irawati Kulkarni, Abhishek Nanda, and Pushpak Bhattacharyya. 2010. Introducing Sanskrit wordnet. In *Proceedings on the 5th global wordnet conference (GWC 2010)*, pages 287–294, Mumbai.

Kuncoro, Adhiguna, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia.

Lample, Guillaume, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of ofNAACL-HLT*, pages 260–270. DOI: https://doi.org/10.18653/v1/N16-1030

Lao, Ni and William W. Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1):53–67. DOI: https://doi.org/10.1007/s10994-010-5205-8

Lapata, Mirella. 2003. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 545–552, Sapporo. DOI: https://doi.org/10.3115/1075096.1075165

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. DOI: https://doi.org/10.1109/5.726791

LeCun, Yann, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu-Jie Huang. 2006. A tutorial on energy-based learning. In Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan, editors, *Predicting Structured Data*. MIT Press, pages 191–241.

LeCun, Yann, Sumit Chopra, Marc'Aurelio Ranzato, and Fu-Jie Huang. 2007. Energy-based models in document recognition and computer vision. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, pages 337–341, Curitiba. DOI: https://doi.org/10.1109/ICDAR.2007.4378728

Liu, Yijia, Yue Zhang, Wanxiang Che, and Bing Qin. 2015. Transition-based syntactic linearization. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 113–122, Denver, CO. DOI: https://doi.org/10.3115/v1/N15-1012

Malaviya, Chaitanya, Matthew R. Gormley, and Graham Neubig. 2018. Neural factor graph models for cross-lingual morphological tagging. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2653–2663. DOI: https://doi.org/10.18653/v1/P18-1247

Matthews, P. H. 2007. *The Concise Oxford Dictionary of Linguistics*, Oxford Paperback Reference, Oxford University Press.

McCarthy, Arya D., Ekaterina Vylomova, Shijie Wu, Chaitanya Malaviya, Lawrence Wolf-Sonkin, Garrett Nicolai, Christo Kirov, Miikka Silfverberg, Sabrina J. Mielke, Jeffrey Heinz, Ryan Cotterell, and Mans Hulden. 2019. The SIGMORPHON 2019 shared task: Morphological analysis in context and cross-lingual transfer for inflection. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 229–244, Florence. https://www.aclweb.org/anthology/W19-4226

McDonald, Ryan, Fernando Pereira, Seth Kulick, Scott Winters, Yang Jin, and Pete White. 2005a. Simple algorithms for complex relation extraction with applications to biomedical IE. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 491–498.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver. DOI: https://doi.org/10.3115/1220575.1220641

Melnad, Keshav S., Pawan Goyal, and Peter Scharf. 2015. Meter identification of Sanskrit verse. In *Sanskrit Syntax*, Sanskrit Library, Université Paris Diderot, pages 325–346.

Meng, Changping, Reynold Cheng, Silviu Maniu, Pierre Senellart, and Wangda Zhang. 2015. Discovering meta-paths in large heterogeneous information networks. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 754–764, Geneva. DOI: https://doi.org/10.1145/2736277.2741123, PMCID: PMC4632928

Mittal, Vipul. 2010. Automatic Sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 85–90.

Mohri, Mehryar. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

More, Amir. 2016. Joint morpho-syntactic processing of morphologically rich languages in a transition-based framework. Master's thesis. The Interdisciplinary Center, Herzliya, Israel.

More, Amir, Amit Seker, Victoria Basmova, and Reut Tsarfaty. 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. DOI: https://doi.org/10.1162/tacl_a_00253

More, Amir and Reut Tsarfaty. 2016. Data-driven morphological analysis and disambiguation for morphologically rich languages and universal dependencies. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 337–348, Osaka.

Müller, Andreas C. and Sven Behnke. 2014. Pystruct: Learning structured prediction in Python. *Journal of Machine Learning Research*, 15(1):2055–2060.

Müller, Thomas, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332.

Nair, Sivaja S. and Amba Kulkarni. 2010. The knowledge structure in Amarakośa. In *Sanskrit Computational Linguistics*, pages 173–189, Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-17528-2_13

Natarajan, Abhiram and Eugene Charniak. 2011. S3-statistical saṃdhi splitting. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, pages 301–308.

Nichols, Johanna. 1986. Head-marking and dependent-marking grammar. *Language*, 62(1):56–119. DOI: https://doi.org/10.1353/lan.1986.0014

Nivre, Joakim, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen et al. 2018. Universal dependencies 2.3. LINDAT /CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University, http://hdl.handle.net/11234/1-2895.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, PA. DOI: https://doi.org/10.3115/1073083.1073135

Pollock, Sheldon. 2003. Sanskrit literary culture from the inside out. In *Literary Cultures in History: Reconstructions from South Asia*. University of California Press, pages 39–130. DOI: https://doi.org/10.1525/9780520926738

Puduppully, Ratish, Yue Zhang, and Manish Shrivastava. 2017. Transition-based deep input linearization. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 643–654, Valencia. DOI: https://doi.org/10.18653/v1/E17-1061

Qi, Peng, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels.

Qi, Peng, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108. DOI: https://doi.org/10.18653/v1.18653/v1/2020.acl-demos.14

Ramkrishnamacharyulu, K. V. 2009. Annotating Sanskrit texts based on śābdabodha systems. In *International Sanskrit Computational Linguistics Symposium*, pages 26–39. DOI: https://doi.org/10.1007/978-3-540-93885-9_3

Ratliff, Nathan D., J. Andrew Bagnell, and Martin A. Zinkevich. 2007. (online) Subgradient methods for structured prediction. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*, pages 380–387, JMLR.org, San Juan.

Reddy, Vikas, Amrith Krishna, Vishnu Sharma, Prateek Gupta, Vineeth M. R., and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, pages 1666–1671, European Language Resources Association (ELRA), Miyazaki.

Rother, Carsten, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. 2007. Optimizing binary MRFS via extended roof duality. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, pages 1–8, Minneapolis, MN. DOI: https://doi.org/10.1109/CVPR.2007.383203

Sahin, Gozde Gul and Mark Steedman. 2018. Data augmentation via dependency tree morphing for low-resource languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5004–5009, Brussels. DOI: https://doi.org/10.18653/v1/D18-1545

Scharf, Peter, Anuja Ajotikar, Sampada Savardekar, and Pawan Goyal. 2015. Distinctive features of poetic syntax preliminary results. In *Sanskrit Syntax*. pages 305–324.

Scharf, Peter M. 2013. Linguistics in India. *The Oxford Handbook of the History of Linguistics*. pages 227–257. DOI: https://doi.org/10.1093/oxfordhb/9780199585847.013.0012

Schaufele, Steven William. 1991. *Free Word-Order Syntax: The Challenge from Vedic Sanskrit to Contemporary Formal Syntactic Theory*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Schmaltz, Allen, Alexander M. Rush, and Stuart Shieber. 2016. Word ordering without syntax. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2319–2324, Austin, TX. DOI: https://doi.org/10.18653/v1/D16-1255

Seeker, Wolfgang and Özlem Çetinoğlu. 2015. A graph-based lattice dependency

parser for joint morphological segmentation and syntactic analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373. DOI: https://doi.org/10.1162/tacl_a_00144

Seeker, Wolfgang and Jonas Kuhn. 2013. Morphological and syntactic case in statistical dependency parsing. *Computational Linguistics*, 39(1):23–55.

Seker, Amit, Amir More, and Reut Tsarfaty. 2018. Universal morpho-syntactic parsing and the contribution of lexica: Analyzing the ONLP lab submission to the CoNLL 2018 shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 208–215, Brussels.

Shao, Yan, Christian Hardmeier, and Joakim Nivre. 2018. Universal word segmentation: Implementation and interpretation. *Transactions of the Association for Computational Linguistics*, 6:421–435. DOI: https://doi.org/10.1162/tacl_a_00033

Shi, Chuan, Yitong Li, Jiawei Zhang, Yizhou Sun, and S. Yu Philip. 2017. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37. DOI: https://doi.org/10.1109/TKDE.2016.2598561

Smith, Noah A., David A. Smith, and Roy W. Tromble. 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. DOI: https://doi.org/10.3115/1220575.1220635

Socher, Richard, Christopher D. Manning, and Andrew Y. Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9.

Sproat, Richard, Chilin Shih, William Gale, and Nancy Chang. 1994. A stochastic finite-state word-segmentation algorithm for Chinese. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 66–73, Las Cruces, NM. DOI: https://doi.org/10.3115/981732.981742

Sproat, Richard W., Chilin Shih, William Gale, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404.

Srikumar, Vivek. 2017. An algebra for feature extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1891–1900, Vancouver.

Śrīnivāsa Aiyaṅkār. 1910. *The Ramayana of Valmiki*, Madras : Little Flower Co. Book.

Staal, F. 2008. *Discovering the Vedas: Origins, Mantras, Rituals, Insights*, Penguin books.

Staal, Johan Frederik. 1967. *Word Order in Sanskrit and Universal Grammar*, Foundations of Language Supplementary Series, 5, Springer Science & Business Media. DOI: https://doi.org/10.1007/978-94-010-9947-9, PMID: 5586986

Straka, Milan and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver. DOI: https://doi.org/10.18653/v1/K17-3009

Straka, Milan, Jana Straková, and Jan Hajic. 2019. UDPipe at SIGMORPHON 2019: Contextualized embeddings, regularization with morphological categories, corpora merging. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 95–103, Florence. DOI: https://doi.org/10.18653/v1/W19-4212

Straková, Jana, Milan Straka, and Jan Hajič. 2014. Open-source tools for morphology, lemmatization, POS tagging and named entity recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, MD. DOI: https://doi.org/10.3115/v1/P14-5003

Sun, Weiwei. 2010. Word-based and character-based word segmentation models: Comparison and combination. In *COLING 2010: Posters*, pages 1211–1219, Beijing.

Sutton, Charles, Andrew McCallum, and Khashayar Rohanimanesh. 2007. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8(Mar):693–723.

Takahashi, Hiromitsu. 1980. An approximate solution for Steiner problem in graphs. *Mathamatica Japonica*, 24(6):573–577.

Taskar, Ben, Carlos Guestrin, and Daphne Koller. 2003. Max-margin Markov networks. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 25–32, Cambridge, MA.

Tkachenko, Alexander and Kairit Sirts. 2018. Modeling composite labels for neural morphological tagging. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 368–379. DOI: https://doi.org/10.18653/v1/K18-1036

Tomita, Etsuji, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42. DOI: https://doi.org/10.1016/j.tcs.2006.06.015

Tsarfaty, Reut. 2006. Integrated morphological and syntactic disambiguation for modern Hebrew. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, COLING ACL '06, pages 49–54, Stroudsburg, PA. DOI: https://doi.org/10.3115/1557856.1557867

Tsarfaty, Reut, Khalil Sima'an, and Remko Scha. 2009. An alternative to head-driven approaches for parsing a (relatively) free word-order language. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 842–851. DOI: https://doi.org/10.3115/1699571.1699622

Tubb, Garry and Emery Boose. 2007. *Scholastic Sanskrit: A Handbook for Students*, Treasury of the Indic sciences, American Institute of Buddhist Studies, Center for Buddhist Studies and Tibet House, Columbia University.

Vickrey, David and Daphne Koller. 2008. Sentence simplification for semantic role labeling. In *Proceedings of ACL-08: HLT*, pages 344–352, Columbus, OH.

Voss, Stefan. 1993. Worst-case performance of some heuristics for Steiner's problem in directed graphs. *Information Processing Letters*, 48(2):99–105. DOI: https://doi.org/10.1016/0020-0190(93)90185-C

Wang, Mengqiu, Rob Voigt, and Christopher D. Manning. 2014. Two knives cut better than one: Chinese word segmentation with dual decomposition. In *ACL (2)*, pages 193–198. DOI: https://doi.org/10.3115/v1/P14-2032

Wang, Wenhui, Baobao Chang, and Mairgup Mansur. 2018. Improved dependency parsing using implicit word connections learned from unlabeled data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2857–2863. DOI: https://doi.org/10.18653/v1/D18-1311

Weir, David, Julie Weeds, Jeremy Reffin, and Thomas Kober. 2016. Aligning packed dependency trees: A theory of composition for distributional semantics. *Computational Linguistics*, 42(4):727–761. DOI: https://doi.org/10.1162/COLI_a_00265

Wiseman, Sam and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, TX. DOI: https://doi.org/10.18653/v1/D16-1137

Wolf, J. and W. Woods. 1977. The HWIM speech understanding system. *ICASSP '77. IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2, 784–787.

Xue, Nianwen. 2003. Chinese word segmentation as character tagging. *Computational Linguistics and Chinese Language Processing*, 8(1):29–48.

Yang, Jie, Yue Zhang, and Shuailong Liang. 2019. Subword encoding in lattice LSTM for Chinese word segmentation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2720–2725, Minneapolis, MN.

Zeman, Daniel, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels.

Zhang, Xiang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657.

Zhang, Yue. 2013. Partial-tree linearization: Generalized word ordering for text synthesis. In *IJCAI*, pages 2232–2238. DOI: https://doi.org/10.1162/coli_a_00037

Zhang, Yue and Stephen Clark. 2011. Syntactic processing using the generalized

perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Zhang, Yue and Stephen Clark. 2015. Discriminative syntax-based word ordering for text generation. *Computational Linguistics*, 41(3):503–538. DOI: https://doi.org/10.1162/COLI_a_00229

Zhang, Yue and Jie Yang. 2018. Chinese NER using lattice LSTM. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1554–1564, Melbourne. DOI: https://doi.org/10.18653/v1/P18-1144