A Dictionary Data Processing Environment and Its Application in Algorithmic Processing of Pali Dictionary Data for Future NLP Tasks

Dipl. Inf. Jürgen Knauth
Trier Center for Digital Humanities
Universitätsring 15
54296 Trier
Germany
knauth@uni-trier.de

David Alfter
Trier Center for Digital Humanities
Bollwerkstrasse 10
54290 Trier
Germany
s2daalft@uni-trier.de

Abstract

This paper presents a highly flexible infrastructure for processing digitized dictionaries and that can be used to build NLP tools in the future. This infrastructure is especially suitable for low resource languages where some digitized information is available but not (yet) suitable for algorithmic use. It allows researchers to do at least some processing in an algorithmic way using the full power of the C# programming language, reducing the effort of manual editing of the data. To test this in practice, the paper describes the processing steps taken by making use of this infrastructure in order to identify word classes and cross references in the dictionary of Pali in the context of the SeNeReKo project. We also conduct an experiment to make use of this data and show the importance of the dictionary. This paper presents the experiences and results of the selected approach.

1 Introduction

Pali (also written Pāli, Paḷi or Pāḷi) is a dead language from the group of Middle Indo-Aryan languages (Burrow, 1955: 2). Despite its status as dead language, Pali is still widely studied because many of the early Buddhist scriptures were written in Pali (Bloch, 1970: 8). It is also said that Buddha himself spoke Pali or a closely related dialect (Pali Text Society; Thera, 1953: 9).

SeNeReKo is a joint research project of the Trier Center for Digital Humanities (TCDH) and the Center of Religious Studies in Bochum (CERES), Germany. This project aims to process the Pali Canon – which at the same time is the only texts left of Pali – in order to research religious contacts between the early Buddhists and other religious groups and cultures.

To achieve this we aim to develop NLP tools and process this data as we believe that the concepts of interest will be found in direct verbal expressions within this corpus. From the information we aim to extract we intend to create networks that allow analysis of these concept.

Until now such an attempt has never been made. Even processing Pali using computer algorithms has not been in the focus of the scientific community yet. As we researchers in SeNeReKo try to change this we now focus on a basic building block for NLP tools: Building a machine readable dictionary that allows building sophisticated NLP tools in the long run. To attempt this a digitized copy of the dictionary of William and Davids (1997) has been provided to our team by the University of Chicago.

2 Related Work

As Pali is a low resource language not much work has yet been done in this field, especially not with the dictionary data. The only researchers we know of that have tried to use this data is a team of the

Proceedings of the 5th Workshop on South and Southeast Asian NLP, 25th International Conference on Computational Linguistics, pages 65–73,
Dublin, Ireland, August 23-29 2014.

University of Copenhagen. Their goal was to create a new digitized version of this dictionary. Unfortunately they did not succeed and stopped after having edited three letters of the Pali alphabet. To our knowledge we are the first to work with this data again.

With good success a language somehow similar to Pali has been addressed in the past: Sanskrit (Hellwig 2009). Nevertheless attempts to adapt these tools to Pali have not been possible due to the lack of a suitable dictionary.

Regarding NLP tools addressing Pali some experiments have already been performed by the members of the SeNeReKo project team and especially by David Alfter. Nevertheless no work could yet reach a state of publication due to the lack of a suitable digital dictionary that would serve as a basis for NLP tasks.

3 Technical infrastructure

As it is the nature of digital humanities projects like SeNeReKo a variety of researchers is involved into the process of processing and editing data and developing methods for the research intended. In SeNeReKo this involves Pali experts, Sociologists, Computer Linguists and Scientists (and Egyptologists for performing work with other text corpora not addressed by this paper.) An infrastructure that aims at enabling collaboration is therefore mandatory. This section describes key aspects of the infrastructure developed.

3.1 Dictionary Server

Each dictionary entry is to be understood as a single document which is self-contained and structured. A dictionary is considered to be a collection of documents.

Being self-contained all information relevant to each individual entry is stored in the same document. Each of these entries must be structured to provide information in a clearly defined way for NLP tools in the future.

To store the dictionary data a MongoDB data base is used. This NoSQL data base not only supports such kind of data model it also provides the necessary flexibility to define and change the internal structure of such dictionary document in the future as needed.

For ease of use a NodeJS-based dictionary server has been implemented that provides user authentication and high level data base operations addressing searching, inserting, updating and deleting specific to the requirements of a dictionary.

The pairing of NodeJS and MongoDB is reasonable because of performance reasons: MongoDB receives and returns data not in XML, but in JSON notation; and as NodeJS provides its functionality through a highly efficient JavaScript engine JSON data can directly be processed without any need of conversion.

For collaboration purposes a REST-API has been implemented with compatibility and interoperability in mind. As we aim for algorithmic processing of data and want to enable researchers to easily implement custom NLP tools that make use of the dictionary data independently from each other. To support this as best as possible a Java and C# library has been implemented as well as an R module for convenience.

As it is the nature of dictionary data to consist of a larger amount of individual entries, classical request-response communication models, as they would be imposed by HTTP, are unsuitable for processing (in the sense of algorithm based editing). Following that approach would result in notable performance degradation. Fortunately single processing steps as we intend them for pattern matching and enriching of dictionary entries have largely no relation between individual entries. Therefore the dictionary server provides an interface for bulk communication: A large amount of individual protocol function calls can be packed into a single package. As the server processes them in parallel and returns the response to all requests again in a single response we are capable of overcoming the problem of summation of network latencies and end up with good performance in updating data.

3.2 Data Processing Tool

In SeNeReKo we need to process the original - near plaintext - dictionary entries. This data is inserted into the dictionary server beforehand and then various analysing and processing steps need to be taken. To perform these, we implemented a processing environment that makes developing of individual

processing units very easy, gives high performance and great transparency about data modifications intended by these units.

Our data processing tool is a programming environment for creating small processing units in C#. Data management issues do not need to be addressed: This is done by the programming environment automatically. The individual units are compiled to native .Net code for speed of processing. On execution data from the dictionary server is retrieved and passed through these units and – if necessary – sent back to the server after modifications have been applied. Together with the bulk processing supported by the dictionary server the compilation of the code units speeds up any processing. By directly making use of C# this approach we achieve great flexibility: It allows making use of all kinds of existing libraries if desired and enables researchers to implement all kinds of data specific pattern matching and processing for research tasks.

As it is the nature of dictionary data to consist of a large amount of individual entries, applying pattern matching and transformation tasks require a great deal of transparency. Researchers performing these tasks need to be able to identify which rule is applied to which entry in what form and see what modification an entry will receive. To achieve this transparency our data processing tool collects information about all modifications applied to each individual data record and presents them in a large list that can be filtered by some criteria. Thus our tool aids in debugging by allowing insight into every details of the tasks a researcher is going to perform.

4 Processing of Pali Dictionary Data

Prior to any processing we converted the original digitized dictionary entries we received from the University of Chicago into JSON data structures and inserted them into our dictionary server. In the next sections we present our processing steps applied to the individual dictionary data records within the infrastructure described above.

4.1 Transliteration of Lemmas

As it turned out the digitized version of the Pali Dictionary we received was not entirely in accordance with the current transliteration conventions. Therefore to be able to use the Pali dictionary for research the lemmas had to be adjusted.

To achieve a valid transformation we first had to verify that no accidental errors had been introduced by the original digitization process done by the Pali Text Society. We therefore implemented an alphabet model that follows the old transliteration schema used to represent glyphs of the Sinhalese alphabet. For these single letters one or two Latin ligatures (with diacritics) are used today. Modelling each word with the original alphabet is mandatory to be able to identify possible errors. We checked all lemmata against our model and were able to identify 14 of 16280 lemmata violating our model. The errors could be identified to be printing errors or misinterpretation during digitalization and were then corrected manually before continuing processing.

The next step was to perform substitutions of the letters ‘ŋ’. To ensure correct processing this was not done on the Unicode based character representation of the data directly but on the original letters modelled by our alphabet model. Substitution is performed on that basis taking the phonetic context into account as necessary:

```
ŋ followed by j, c, h or e => ñ
ŋ followed by k or kh => ṅ
ŋ followed by d, dh or n => n
ŋ followed by m, p, bh or b => m
ŋ followed by s => ṃ
ŋ followed by ṭ, ṭh => ṇ
ŋ followed by l => l
ŋ followed by v, y or r => ṃ
ŋ followed by a, e, i, o, u, ā, ī, ū => ṃ
ŋ not followed by any character => ṃ
```

5 Pattern Recognition and Enriching Dictionary Entries

5.1 Pattern Matcher

In processing Pali we had to take our own pattern matching approach in order to avoid problems encountered with regular expressions in C#. We found that some Pali specific diacritics did not get processed as the official regular expression syntax specification suggested. To overcome these limitations we implemented an own pattern matcher.

Nevertheless we were not interested in dealing with space characters as they do not provide any valuable information to our pattern recognition tasks. And for easy communication with Indologists a pattern syntax was required that would be easy to understand. So these requirements specific to our field of application were taken into account in building the pattern matcher.

The pattern matching system we designed does not process character streams but token streams. The system can distinguish between the following concepts:

- A whitespace – which is automatically left out during tokenizing the dictionary articles
- A word – which is an alphanumeric character including all diacritics
- A delimiter – which is any kind of character not being to a word or whitespace

As we aimed for an iterative process in order to identify relevant pattern it helped greatly to be able to express patterns to be matched in the form of expressions that are easy readable by non-computer experts. Our syntax supports the following forms:

- Match a specific word token
- Match any word token
- Match a specific delimiter token
- Match any delimiter token

Examples of this syntax are given in the next sections which address specific pattern recognition tasks individually.

5.2 Cross References

As Pāli grammar is not standardized to the same extent as, e.g., Sanskrit, various alternative word forms occur. The Pali dictionary at hand addresses this problem to some extent by containing several versions of some lemmas. These entries then contain purely textual information of a reference to the dictionary entry having more information about the selected lemma. In the Pali dictionary this is expressed in forms like this:

```
... in general see <b>buddha<b> ...
```

Such a form is matched by a pattern like this:

```
'in'  'general'  'see'  <  'b'  >  W*!  <  /  'b'  >
```

The pattern specified is easy to understand: This is a sequence of individual patterns matching specific tokens. Words in inverted commas express an exact match of a single word. “`W*!`” indicates that a word of any kind is expected here (and it should be available for further use after a match has been found). Other characters match specific delimiter tokens.

Two real world examples of dictionary entries:

```
anumatta
    see <b>aṇu°</b> .

ano
    is a frequent form of comp<superscript>n.
```

```
</superscript><b>an--ava</b> , see <b>ava</b> .
```

As there exist various different forms of patterns like this in the dictionary specifying multiple possible variants was required. Within an iterative process we were able to identify 46 different kinds of patterns which we could make use of for automatic identification.

To further help manual processing of the dictionary we implemented a verifier that tries to identify the lemmas each cross reference refers to within the dictionary. This is done by direct dictionary lookup. References that do not seem to point to a valid lemma are listed together with candidates based on Levenshtein distance for manual processing later by Indologists.

5.3 Extracting word class information

As we aim for lemmatizing and part of speech tagging of the Pali Canon, in the long run having information about the word class of each lemma is mandatory. Therefore we used pattern matching to aid the generation of data for this purpose.

Our algorithmic approach of classification is basically performed in three steps described next.

Word class information mainly manifests itself in expressions enclosed in rounded brackets. E.g.:

```
apāra
     (nt.) [a + pāra] 1. the near bank of a river ...

sīhaḷa
     Ceylon; (adj.) Singhalese ...

susira
     (adj.--nt.) [Sk. śuṣira] perforated, full
     of holes, hollow ...

pītika
     (--°) (adj.) [fr. pīti] belonging to joy; ...
```

Unfortunately round bracket expressions are used in different semantic contexts within dictionary entries. In a first step we therefore extracted all content enclosed in round brackets and identified expressions that represent word class information. Though an old printed edition of the dictionary contained a clear definition of these word class expressions used we encountered some variety of writing, of combination and of misspelling: Building a list of relevant expressions was the only way to address all phenomena in sufficient quality.

Secondly we know from Pali grammars that verb lemmata typically end with “-ti” in the dictionary. But not all lemmata ending with “-ti” are verbs. Therefore we implemented the following algorithm that was able to clearly identify lemmata correctly as verbs:

```
for all lemmas do
    if lemma does not end with “-ti” -> reject it
    if bracket expression in data matches a pattern clearly
         classifiable as non-verb -> reject it
    if entry does not contain the (English) word “to” -> reject it
    otherwise -> recognize this lemma as being a verb
```

After having identified verbs successfully we then were able to address dictionary entries of other word forms purely according to expressions in round brackets. The following list gives an overview of how many kinds of patterns have been identified and were involved in this process:

Word Class	Number of Patterns
adjective	26 incl. one misspelling
indeclinable	1
adverbs	4 incl. one misspelling

pronouns	1
numerals and ordinals	2
nouns	8

6 Word class recognition

In order to evaluate the importance of the dictionary, we designed the following task: for each word in a manually tagged subset of the Pali Canon, we tried to recognize the word class using a generation-based and a heuristic approach. We then compared the results of both approaches.

For the generation-based approach, we generated all possible word forms, including morphological information, for every word in the dictionary using the morphological generator. The generator uses paradigms to generate regularly inflected word forms. Furthermore, the generator uses the dictionary to look up morphological information about a word and, if present, uses this information to restrict the generation to grammatically adequate forms. However, since the dictionary entries do not always present this information, or because it's not always possible to easily extract this information, we over-generated in cases where no information can be retrieved from the dictionary. We also generated rare forms according to information presented in available grammars on Pali. In total, we were able to generate 11447206 word forms for all words. This averages to about 702 word forms per dictionary entry. In compact notation, this resulted in about 1.5 GByte of data.

As we generated possible morphological forms from lemmas, we then reversed the data structure to arrive at a morphological form lookup table. We saved these results locally for later efficient lookup.

As a test corpus for our word class recognition task we used a manually annotated set of 500 sentences (about 4600 words). These sentences have been extracted earlier in the SeNeReKo project, choosing three consecutive sentences at random from the whole Pali corpus. This preparatory step has been started about a year ago to assist future computational linguistic tasks (a further 500 sentences are work in progress). Thus, the data is representative of the whole corpus and is not biased.

We then stepped through our corpus and checked for each word whether one or more of the generated forms corresponded to the word at hand. If this was the case, we retrieved the relevant entries including all attached morphological information. From these entries, we then retrieved the word class information for the word.

For the heuristic approach, we built a morphological analyzer. The analyzer can only rely on its internal heuristic for guessing the word class of a word. The heuristic is ending based and uses paradigms to determine to which word class a word could belong. The analyzer tries to identify and separate possible endings occurring in different paradigms. Based on these analyses, the word class is guessed.

Before we could start the experiment, we had to map the word classes used by the generator/analyzer and the word classes used in the annotated corpus onto a common set of classes. The reference corpus uses a fine-grained tag set that's standardized for use in more than one corpus in the SeNeReKo project. The dictionary uses a simple tag set, which has been created independently of the SeNeReKo tag set many decades ago. The tag sets follow different principles and goals. It is therefore not always straightforward to map one tag set onto the other.

We tried to assign each word of the reference corpus a word class and checked the results against the manual annotations. The results of this algorithmic output are evaluated in the result section below.

7 Discussion

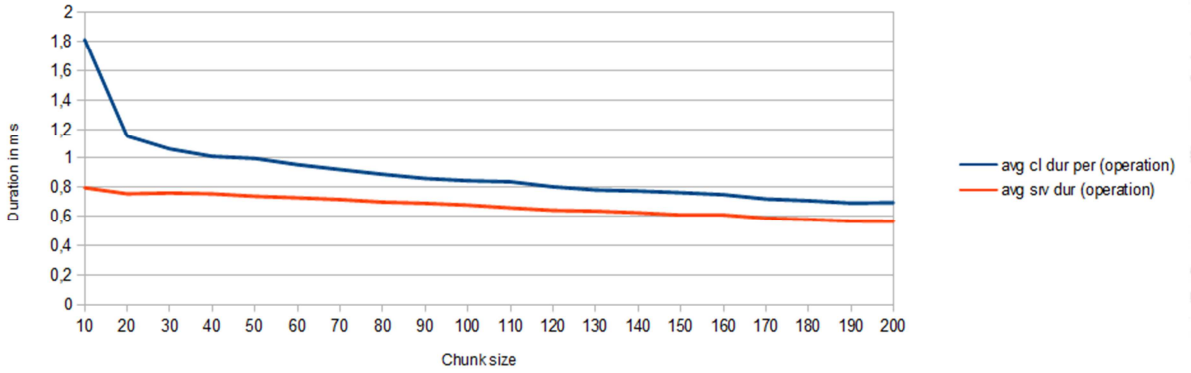
7.1 Performance of the processing environment

As a server we use an older 32 bit Linux machine with an Intel Core Duo at 2.4 GHz and 4 GByte of memory which runs the dictionary server with its data base.

Due to bulk processing of requests we were able to bring down the average time for a single write operation to about 0.7ms per dictionary entry from a client's point of view under ideal circumstances. In a real world application such as our data processing tool this enables us to process all 16280 dictionary entries within about 10 seconds if no changes are applied and to about 20 seconds if all

entries must be read and written back to server. We found this delay very acceptable during our design and implementation of individual processing units for the dictionary data.

The following performance measurement chart for data write requests gives an insight into how performance is affected by network latency:



(If the above chart is displayed in black and white: The top line represents the client duration measured per operation, the bottom line measures the server duration per individual insert operation.)

This measurement is taken by inserting all Pali dictionary data 10 times with different chunk sizes and averaging the duration as measured by the test software. For convenience the server performs performance measurements on his own and sends his results 9together with the response to the client, so that such a kind of analysis can be performed easily. The difference between both measurements indicate the overhead introduced (mainly) by network latencies.

Please note that the chart starts at a chunk size of 10. This is for a reason: It turned out that lower values will introduce significantly more delay.

7.2 Results of pattern matching

Our attempts to process the 16280 dictionary entries resulted in being able to recognize word forms in 10016 of all entries. This is about 61.5% of all dictionary data.

Regarding cross references we were able to extract 457 cross references to existing lemmas within the dictionary, 52 references to lemmas not in the dictionary and 75 references containing only incomplete information and cannot be resolved automatically.

At first hand these values do not seem to be very high. But as we can only rely on clearly identifiable patterns within the dictionary entries these values are even better than we hoped at the beginning of our work. It has been clear right from the start that a greater amount of dictionary entries would need to be the centre of manual work in the future by Pali experts: Many entries simply do not contain any information that can be recognized by the algorithmic approaches taken.

As Pali is a largely dead language we have to consider that our data processing described in this paper is a one-time task. The only relevant dictionary at hand is the one we used, containing exactly those words we have. We successfully identified word classes for lemmas leaving 6264 for manual processing for our Indological colleagues. If even more time would be spent in finding even more patterns within the dictionary entries, we might improve our performance by a few percent, but there is no real reason to do this: We have come to a point where finding more patterns will take considerably more time than identifying word classes and assigning them manually to the dictionary entries.

7.3 Results of Word Class recognition

We tried to recognize word classes based on the generation-based approach and on the heuristic approach as described above. We faced the problem that word forms can be analysed in more than one way, even by using paradigms, which represent regular inflections. This degree of ambiguity cannot be resolved currently due to the particularities of Pali, such as a high degree of homonymy. Furthermore, different paradigms yield the same surface form, even though they belong to totally different word classes.

Therefore, we evaluated the resulting data in two different ways. First, we used “is-any” matching. If a test corpus word has been assigned more than one word class by our algorithms, we consider the

word classes to match if the two sets share at least one common element. This way we address the problem of ambiguities. Second, we used “exact” matching. In this case, we consider the result to be a positive match if and only if the proposed word class corresponds exactly to the assigned word class. By using this approach, we try to determine the degree of unambiguousness with which we can propose a word class. If a word is assigned a word class and the program suggests two word classes, of which one corresponds to the assigned word class, we count this as a failure.

Please note that, since it’s not always possible to distinguish clearly between nouns and adjectives in Pali, we aggregated these word classes into one class. To this class we also counted words tagged as ordinal adjectives, since they are inflected like regular adjectives.

The following tables illustrate our results:

“is any” matching		
	Generation based	Heuristic
Noun-adjective-ordinalAdjective	63.30%	99.96%
Numeral	61.04%	76.62%
Pronoun	82.75%	88.57%
Verb	51.24%	63.37%

As you can gather from the table, the performance of the word form generation based approach did not match the performance of our heuristic approach in the first experiment. Further investigation showed that this is mainly due to the fact that not all necessary word forms encountered in the reference corpus could be generated. There are several reasons for this: First, the exact ways to generate word forms are not yet completely covered by literature and in some areas are still under research: e.g. at least regarding verb forms, there is still ongoing research. Second, our generation process was not able to handle irregular forms well because this information is not yet represented in the dictionary. This data will probably be entered by Pali experts next year. Third, most of the forms we could not recognize are sandhi and other compound forms. This is a task the generation process cannot handle well in general. A heuristic approach does not encounter these problems.

To better judge our algorithms, we therefore evaluated the results only for word forms that could be addressed by these algorithms. The following tables give an overview about these results:

“is any” matching (processable words)		
	Generation based	Heuristic
Noun-adjective-ordinalAdjective	97.31%	99.96%
Numeral	81.03%	76.62%
Pronoun	86.61%	88.57%
Verb	76.25%	63.37%

As you can see, on word forms that could be processed, both approaches work similarly well.

With the current state of the dictionary, these results are as good as can be. Please note that while the heuristic approach must be considered to be final the generation based approach will improve over time as the dictionary will be improved by the Pali experts in the next years.

Our “exact” evaluation operator revealed that word forms in the reference corpus that uniquely belong to a single word class can be recognized much better by the generation based approach than by the heuristic approach. Interestingly, though we are still lacking information about irregular verb forms in the dictionary, we achieved up to 60.37% precision on verbs in exact word class recognition, while the heuristic approach surprisingly did not succeed very well.

The approaches we took can surely be improved. However, these approaches rely heavily on a dictionary, which is more detailed and even more complete. Pali experts will provide this data in the future but this is an ongoing process which will take a few years.

7.4 Conclusion and Future Work

In this paper we have addressed the task of extracting cross references and word class information from dictionary entries in a Pali dictionary. For this task as well as for future computer linguistic tasks, we have built an infrastructure suitable for data management and processing. We have experienced that even if the individual articles are not written in a consistent and clear way, some information still can be extracted. We therefore propose that similar approaches might be taken with dictionaries of other dead languages as well in the future based on the technical infrastructure we created.

We tried to complement our approach with taking the English translations, contained in most of the dictionary entries, into consideration. Unfortunately this did not work well due to the nature of our data: Most of the dictionary entries do contain a discussion of a lemma in English, but as the individual dictionary entries don't follow a clearly defined structure and even discuss various related words within these entries it turned out this approach is too incomplete and too error prone to be usable in practice.

We found the processing environment to be of great help in order to shorten the time consuming manual processing of data. Three aspects we like to point out in this context: The concept of having an integrated development environment that takes data management work off the shoulders of researchers and allows writing small units of code for processing turned out to aid in this process. Furthermore the transparency given by the system about processing details for every single word helps greatly to avoid mistakes and therefore saves time of researchers.

Our experiment concerning word class recognition showed that the dictionary is essential. While the dictionary data is still relatively incomplete, we were able to get good results. Future work needs to be done in this area, especially the correction of lemmas and part of speech tags in the future. However, this is a future task that goes beyond the scope of this paper.

A custom dictionary editor has been built that connects to the dictionary infrastructure at hand. With this tool our Indological collegues intend to perform the unavoidable manual improvement in the next years. If this process is completed at some point in the future we intend to address lemmatizing and part of speech tagging again, something that can not yet been done to a fully satisfying extent right now. Nonetheless, as our word class experiment showed, we were able to achieve good results despite the problems encountered. It is to be expected that with the improvement of the dictionary, the results will also improve in the future.

Reference

Alfter, David. 2014. *Morphological analyzer and generator for Pali.*

Critical Pāli Dictionary. Web.

Collins, Steven. 2006. *A Pali grammar for students.* Chiang Mai: Silkworm Books. Print.

Geiger, Wilhelm. 1943. *A Pali Grammar.* Pali Text Society. Print.

Helwig, Oliver. 2009. *SanskritTagger, a stochastic lexical and POS tagger for Sanskrit.*

Stede, William and Davids , Rhys. 1997. *Pali-English Dictionary*. 2nd ed, Motilal Banarsidass. Print.

Pali Text Society. Web.

Thera, Nārada. 1953. *An elementary Pāḷi course.* 2nd ed. Colombo: Associated Newspapers of Ceylon. BuddhaNet eBooks. Web. N.d.