International Journal of Engineering Research in Computer Science and Engineering Engineering (IJERCSE) Vol 2, Issue 3, March 2015

Survey and Analysis of Computing for Sanskrit related to Text Search and Tutoring Ontologies

Rajitha V.[1],[2] Kasmir Raja S. V,[3] Meenakshi Lakshmanan
[1] Dean – Research, SRM University Chennai 600 005, India
[2] Department of Computer Science, Meenakshi College for Women Chennai 600 024, India
[3] Research Scholar, Mother Teresa Women's University Kodaikanal 624 101, India.

Abstract — There has been a lot of work done in computing for Indian languages over the past decades, and especially so in Sanskrit. The problem of comprehensively searching for words or phrases in Sanskrit text has posed a challenge for decades though its usefulness to a variety of users cannot be overemphasized. The sub-problems of *sandhi*-processing and *vibhakti*-processing in Sanskrit have been dealt with by a few researchers and their work is analyzed here. The authors have developed a new solution to the problems and a comparison of this with earlier work is also presented here. A novel and comprehensive *sandhi*-tutoring ontology has been developed by the authors, and its comparison with existing tutors has also been presented here.

Index Terms — Sanskrit, Text search, ,*sandhi*, *vibhakti*, ontology, comparative study

I. INTRODUCTION

Mines of information are available in Sanskrit books. These include books on the *Vedā*, the *Upaniṣads, purānās,* epics, grammar, philosophy etc. It spans various field of study, viz. mathematics, science, engineering, medicine,politics, *Ayurvedā*, spirituality etc. These books do not have any extensive table of contents or word indices and searching in these books hence is being done manually. Manual search is cumbersome because to search a particular word or phrase one has to go through the entire book. We could hence imagine how tedious and time consuming this would be specially when there are hundreds and thousands of verses in the text to be searched like in the *Ramayana* or *Mahabharata.*

The need and significance of word search in Sanskrit documents

Searching in Sanskrit books is a significant problem that has to be addressed because it benefits a lot of people. It is useful to modern researchers, scientists, traditional pundits, teachers etc.A researcher in History might want to find out the period of history a particular work belongs to or author's affiliations etc. A Vedanta teacher might want to search for relevant quotations from different *Upaniṣads* and spiritual texts for teaching the topic under study. A scientist researching in Ayurveda, might want to find details about a particular herb. An artist might want to find details about an *abinaya* in a *NatyaSastra* text.Sanskrit scholars would want to find quotations from different texts for a particular topic for the purpose of scholarly debate. Hence, dating of a text and fixing its authorship with certainty, analyzingthe writing style of an author, finding occurrences of words and studying them in different contexts, quoting from authoritative texts, locating the portion in a text or texts in which a particular usage or word is found, semantic analysis and understanding of texts etc., require comprehensive searching of Sanskrit books.

Searching in Sanskrit E-texts

Most of these sanskrit books are available as e-texts in various digital libraries. Göttingen Register of Electronic Texts in Indian Languages (Germany), The Sanskrit Library (US),Clay Sanskrit Library (UK), Digital South Asia Library (University of Chicago), SARIT (Austria), Mukthabodha Digital Library (Delhi), Digital Library of India (IISC, Bangalore)are among the popular digital libraries.

With thesanskrit e- texts in place, searching in sanskrit documents would seem a trivial task.Searching in Sanskrit texts is not a trivial problem.

Searching in Sanskrit documents is a very complex task. A basic search for the word as is will not be sufficient.The given search word might not be directly present in the text but in a transformed form, owing to grammatical transformations caused by *sandhi* and *vibhakti*. Owing to*sandhi* a search word for example, *gurubhayaḥ*might be found in the text as *gurubhyas* or *gurubhyo*.Search word *śārṅgin* might be found in the

text as *śārṅgiṁḥ* or *śārṅgiṁs*or *śārṅgiṁś*. The search word *asamardhiḥ*might be found in the text as *āsamardhiḥ, āsamardhis, āsamardhir, asamarddhiḥ, asamardddhiḥ. Sandhi* can cause internal word tranformations or external word transformations, meaning *asamardhiḥ*would change to*āsamardhiḥ*, when the search word coalesces with a preceding word ending in *a* or *ā*(external *sandhi*).*asamardhiḥ*would get transformed to *asamarddhiḥ.* (Internal*Sandhi*).With respect to handling *vibhakti*, a search word *div* might be present in the text as*dyauḥ*, a search word *hari* might be present in the text as *hariṇā.* All these transformations are governed by rules of Pānini. Hence the search word along with its transformed forms (caused by *sandhi* and *vibhakti*) ought to be generatedand searched for a complete and efficient search.All the rules governing *sandhi* and *vibhakti* have to be thoroughly studied and implemented. There are complex precedence rules, exceptions and rule clashes that needs to be handled.

The problem of searching in Sanskrit documents has not been addressed comprehensively and solved before. The new work by the authors hence has unraveled a new problem in its first step and has duly addressed and solved the word search problem bydevelopingcomputational algorithms for comprehensively searching and effectively locating specified words and phrases in Sanskrit e-texts. A search engine hence has been developed which handles *sandhi* and *vibhakti* for a comprehensive and efficient search. Hence this work is unprecedented in literature.

The search problem has to handle *sandhi* and *vibhakti*, because of which the search word might be found in a transformed form in the text. Hence all the transformations caused to the search word due to *sandhi* and *vibhakti* needs to be generated and searched.

Comparative study of work on Sanskrit E-text search

As mentioned above, searching in Sanskrit e-texts comprehensively meaning taking in to consideration both *sandhi* and *vibhakti* has not been done before. To start with a thorough literature survey was done to find any existing search engines. In the prevalent digital libraries mentioned earlier and research that has been carried out over the years in natural language processing with reference to Sanskrit, none of them have a complete comprehensive search mechanism.

1. In GRETIL, Germany[1] an option for searching the e-texts has not been provided.The e-texts can only be downloaded or viewed in the browser.
2. Digital South Asia Library, University of Chicago [2], does not provide a search option.
3. Digital Library of India, IISC Bangalore [3] does not provide a search option.
4. Clay Sanskrit Library, UK [4] has proposed to launch a fully searchable corpus. As of now there is no search option.
5. In Sanskrit library, US[5] a search option has been provided. But it provides onlya basic search option only i.e. searches words as is.
6. In Mukthabodha Digital Library [6], the e-texts were displayed in the browser earlier and the site had provided a basic search engine which searches word as is. Now the e-texts are available for download and the basic search engine has been removed from the site (as on 14 February 2015).
7. In SARIT (Search and retrieval of Indic texts), University of Vienna, Austria[7], a basic search option is provided which searches word as is.
8. Under the guidance of AmbaKulkarni, the Department of Sanskrit Studies, University of Hyderabad has developed a search engine for Sanskrit texts [8]. This search engine does take case-inflections in to consideration for search but does not handle *sandhi.*

   This search engine is based on the morphological analyser and a generator developed by the consortium of 7 institutes led by the Department under the project "Development of Sanskrit computational toolkit and Sanskrit--Hindi Machine Translation system" funded by TDIL programme of DIT (2008-2012).

In a Nutshell, from the above listing it is clear that either there is no search option or there is only a basic search option where a word is found as is or there is a search engine which handles only *vibhakti.*

In novel work done by the authors, a comprehensive search mechanism has been developed which handles both *sandhi* and *vibhakti*. A novel algorithm is developed and implemented for generating transformed forms of the search word based on *sandhi*and *vibhakti.* The method of computation for handling *sandhi* and *vibhakti* in the new work by the authors is detailed in the upcoming sections.With respect to word search, the algorithm is developed to facilitate both fast search and slow search. With respect to case-inflections, the initial algorithm generates all the inflected forms (removing repetitions, example *rāmābhyām* appears thrice in the declension table of word *rāma*). Then optimization of the computational algorithm was done, such that the inflectional forms of words that are needed for a

comprehensive word-search was alone generated and hence it resulted in reduced number of computations (more than 80% efficient compared to the complete case-inflection forms generator).

The optimized algorithm mentioned above, enables fast searching. When we search for the masculine word rāma, the algorithm generates the alternative forms such as rāmā, rāme, rāmau and rāmai and searches for instances of these words. The result of search would also result in words such as 'rāmābhiḥ' which is feminine word, if present in the search text. This is a fast search, because the number of computations is around 80% less [9] and the user can ignore the words they are not interested in.

Looking at the example above, searching for a masculine word, also lists feminine words as part of the search result. Similarly while searching feminine words the search result might show up masculine words. For example, if the user searches for the feminine word '*rāmā*', then the search would yield instances such as '*rāmeṇa*' and '*rāmebhyaḥ*', which are inflected forms of the masculine word '*rāma*'. Hence, the resultant words of fast search might not be relevant and meaningful. If the user wants a more meaningful search result, then the algorithm that generates all unique inflected forms of the given word taking into account the word's gender can be used. Since this involves more computations, there would be a small time lag for the program to actually produce the search results, but meaningful search results are obtained. At any rate, the method of computation has been simplified to such an extent that this search does not cause very visible time lags.

There is another aspect that has to be handled for the search to yield truly meaningful results as desired. Giving the search word as '*rāma*, masculine', yields the forms '*rāmaḥ*', '*rāmau*', '*rāmāḥ*', '*rāmebhyaḥ*', '*rāmeṣu*', etc. Now the word '*rāmebhyaḥ*' (variously meaning for/from the many *Rāmas*) might occur in the text as '*rāmebhyo*' as a result of a euphonic conjunction (*sandhi*) with a word that follows it in the text. The elaborate algorithm would not recognize this instance. Similarly, '*rāmau*' could appear in the text as '*rāmāv*' again as a result of a *sandhi* operation. This too would go unnoticed by the algorithm. To solve this problem, we pass all the complete forms of the given word of a given gender into the *sandhi* engine developed [10]. This *sandhi* engine generates those forms of each of these words that arise when the word conjoins euphonically with possible words adjacent to it. This slows down the searching process, but eliminates many irrelevant instances.

The above discussion brings to light a new methodology adopted for searching which has not been thought of before.Even though the University of Hyderabad has developed a search engine,

- Firstly, it does not generate transformed forms of the search word resulting due to *sandhi*.
- Secondly, it does not optimize the search algorithm and does not generate inflectional forms of words that are needed for a comprehensive word-search alone there by enabling fast search. It generates the complete declension table.
- Finally, does not send the results of the declension engine to the *sandhi* engine, to check if further *sandhi* processing is required as discussed above.

Comparative study of work on *sandhi*processing

Advanced Heuristic Algorithm for *Sandhi* Processing (*Sandhi* Joiner) is proposed in [11] by Ravi pal.The algorithm uses a *sandhi* rule base. The algorithm reads rules from the rule base, applies each of the rules to the input words and if a match is found, the resultant word formed is displayed. As the title claims the heuristics defined for the algorithm have not been presented clearly. As stated in the conclusion of the paper, all types of *sandhi* have not been handled.

As part of a morphology generator, a*sandhi* synthesis module was developed by Gerard Huetand presented at the XIth Sanskrit International Conference in Torino in April 2000. Theoperations were specified as tables, defined by information collected from various Western grammars [12] and were not completely based on the traditional approach.

Gerard Huet has modeled Sanskrit processing using finite state automata [13][14].The automaton models 200000 states in to a shared structure of 7500 nodes, with reference to *sandhi* analyzer[13]. Large FSM with many states and transitions can be difficult to manage and maintain.

Gerard Huet proposed a methodology for Sanskrit processing with the finite-state transducers toolkit. *Sandhi* computation here is modeled usinga two-tape transducer [15][16]. It is stated thatinternal *sandhi* is not open to finite state techniques [17].

Malcolm D. Hyman, has also used FSM to model *sandhi* operation [18]. Here only the core rules of external *sandhi* have been modeled and internal *sandhi*'s have not been dealt with.

In the POS tagger [19] by Oliver Hellwig, *Sandhi* processing was carried out as a look-up strategy where *sandhi* rule base is stored in a trie data structure and the Sanskrit tagger is a stochastic tagger which performs part-of-speech tagging with Hidden Markov model.

AmbaKulkarni states that FSAs [20] are the efficient means to model morphology. She has used readymade databases for quickly developing Sanskrit morphological analyser. The existing resources provided all the analysed forms, from which a Finite State Transducer [21] was built. The system used the lt-toolbox14 – a tool developed by the Apertium group for developing the FST for Sanskrit processing.

As part of the work done by the authors, around 100 Pāṇini*sūtras* relevant to *Sandhi*have been codified in a novel way and the *sandhi* engine developed has been thoroughly tested [25] and hence is error free.

In a nutshell, the existing*sandhi* engines have the following characteristics:

- They are not completely based on traditional approach (Pāṇini Grammar)
- They are modeledusing Finite State Automata, more the number of states, more the number of nodes and state transitions, hence it is difficult to manage and maintain.
- As the number of states and transitions grows, the more spaghetti-like the code becomes.
- Some are implemented using heuristics and stochastic models. In stochastic modeling, the successive states of the system is obtained probabilistically hence may not be predicted precisely. In the new work by the authors, a computational algorithm has been developed.The algorithm is deterministic and is a computational algorithm unlike FSA or heuristic modeling.The predictable nature of FSAs are not required for the current problem because the algorithm has been designed in such a manner that the execution proceeds from top to bottom. This has been possible because of the unique method adopted for ordering the Sūtras.
- Modeled using FSA, involves back tracking [13] but the new work by the authorsdoes not involve backtracking.
- Uses table look ups and databases to store rules.
- They are not comprehensive, does not cover the complete gamut ofPāṇini*sūtras*relevant to *Sandhi*.
- Not much attention is given to ordering of *sūtras* which is very important for efficient processing
- Has limitations for example, does not handle internal *sandhi*, doesn't give correct output for some inputs.
- Pāṇini*sūtras* not displayed, not even common name of the *sandhi*
- The Paṇiṇi*sūtra* that is triggered has not been shown
- Most of the *sandhi* rules are mandatory rules, some of them are optional rules and some of them do not bring about any transformation i.e. no-change rules. This aspect has not been handled in the work mentioned above. The new work done by the authors handles this aspect.
- Doubling rules are not handled by the earlier work.

As it is clear from the above discussions, there are shortcomings in the existing *sandhi* engines. The following explains the features of the work done by the authors:

i. This work has for its basis, the *Siddhānta-kaumudī* [26] and the *Kāśikā* [27]. Both are recognized by scholars as authentic and unabridged commentaries on Pāṇini's work.

ii. A computational algorithmhas been developed. The work is based on a unique binary schema [28] that simplifies rule codification to minimal binary set-unset operation. The scheme is binary, and hence lends itself to speedy processing. Being obvious that binary operations are done at the implementation level, the new work by the authors provides a novel binary schema at the representation level also hence it offers itself to speedy processing. The rule simplification is itself the testimony for the efficiency of the binary schema. This schema paves way for rule simplification and hence provides a simple solution to a complex problem.

iii. Very importantly, the existing *sandhi* processors do not lend attention to ordering the *sūtras*. The new work by the authors the Pāṇini*sūtras* have been ordered in a novel way such that the execution is from top to bottom and no need for back tracking, output of one sūtra will be input to *sūtras* that come later and not before. Hence no backtracking is involved. After an in-depth study, the ordering of *sūtras* has been done as follows:
    - Set 1 contains relevant aphorisms of the *sapādasaptādhyāyī* and few specific rules from the *tripādī*
    - Set 2 - Remaining relevant aphorisms of the *tripādī*

- Rules in Set 1 are parsed in reverse order of their *Aṣṭādhyāyī* order.
- Rules in Set 2 are parsed in the *Aṣṭādhyāyī* order.
- No rule already parsed has to be parsed again so, the flow of the program proceeds from top to bottom
- Ordering is changed to accommodate certain rules to parse them before the main rule.
- Few rare rules of Set 2 have to be processed before Set 1.
- In Set 2, all rules that form exceptions to a particular rule are stated after it by Pāṇini, but clearly, have to be processed before the rule by the algorithm
- Before visiting a rule, overall conditions are checked, thus increasing the efficiency of the algorithm

iv. The complete set of *sūtras* relevant to *sandhi* has been codified. Over 100 *Sandhi* aphorisms of Pānini have been formulated in a simplified way.

v. The implementation is error free, takes in to thought, mandatory rules, optional rules and no change rules. It works correctly for internal *sandhi* also, as shown earlier.The unique ordering of rules takes care of rule precedences, rule clashes and exceptions.

vi. Therefore, from the above discussion it is clear thatthe new work by the authors is faster than using a heuristic approach or FSA or stochastic modeling.

Comparative study of work on *sandhi*-tutoring

The traditional and modern methods of studying about euphonic conjunctions in Sanskrit follow different methodologies.

a. Traditional Approach - involves a rigorous study of the Pāṇini'sĀṣtādhyāyī and not suitable for beginners

b. Modern Approach - involve the study of an important *sandhi* rules with the use of examples, one doesnot gain comprehensive understanding of how the rules operate.

This is because there are numerous *sandhi* rules and exceptions,and also complex precedence rules.Knowledge on ordering of *sandhi* rules is not gained. Nuances with respect to the application of *sandhi* rules are not grasped from the existing approaches. Hence developing a new ontology for *sandhi*-tutoring was the need of the hour. The new work by the authors presents a comprehensive ontology designed to enable a student-user to learn in stages all about euphonic conjunctions and the relevant aphorisms of Sanskrit grammar and to test and evaluate the progress of the student-user. The *sūtras* are meaningfully modularized in the ontology.The ordering of the *sūtras* is maintained in such a way that only the knowledge gained in previously covered modules are used in later modules. This is the unique feature of the ontology. The ontology forms the basis of a multimedia *sandhi* tutor that was given to different categories of users including Sanskrit scholars for extensive testing. Such an ontology developed for comprehensively tutoring euphonic conjunctions is the first of its kind.

Comparative study of work on *vibhakti*processing

Vibhakti in Sanskrit has eight cases and three numbers (singular, dual and plural), thus giving rise to a total of 24 case-inflectional forms. There are three genders for nouns in Sanskrit - masculine, feminine and neuter. Case-inflections are defined based on the gender of the word. Different rules are formulated based on the last letter of the word. Example - rāma (masculine ending in the letter a), hari (masculine ending in the letter i). Different rules are formulated for subspecies of words of a particular gender ending in a specific letter. Example - dātṛ (masculine ending in the letter ṛ) and bhrātṛ (masculine ending in the letter ṛ). Nouns of all the three genders dealt in two categories: Ordinary (sādhāraṇa-śabda) and Special (viśeṣa-śabda).

Sandhi processing is involved in the declension engine [22] developed by Gerard Huet. The declension engine operates with around 86 tables each of which describes 24 combinations of 8 cases and 3 numbers). The inflectional forms are generated using *sandhi* operation [15]. The morphology laws for flexion are maintained as paradigm tables. For instance, for nouns, adjectives and pronouns, each table maps a pair (number, case) into a set of possible suffixes.

GirishNathJha et al [23] have developed and implemented the Sanskrit morphological analyser using database models. They store the suffixes and stems in the databases.Sanskrit Subanta Generator was developed by GirishNathJha. The algorithm applies *sup* rules of Pāṇini to the base word and sends the result to *sandhi* processing. Hence it requires *sandhi* processing for generating each of the 21 (7 * 3) case affixes. It also uses a database of nominal bases.

A *SubantaPada* Analyzer for Sanskrit[40] is developed by SmitaSelotet. al. This work identifies subantapada by word splitting. The suffixes are stored in the

database. A numbering scheme is used for the suffixes. But there are issues

a. The paper mentions about *sandhi* analysis but has not presented the complete algorithm and on what authoritative basis the algorithm works.
b. In table 1, numbering scheme not mentioned for dative case
c. The range for numbering *vibhakti* is mentioned as 1-3 in Fig 2, but in table 1 number from 1 – 6 have been used.
d. Four digit numbering scheme is used for noun suffixes, but Fig 2 mentions five digit suffix being used.

Semusi [41] (Sanskrit noun generation and analysis), is a subanta generator, being developed by the Academy of Sanskrit Research, Melkote.

With all this in place and taking in to consideration the complexity of the problem such as in the example below

- simple search for the word rājan in a text would not yield the forms rājā or rājñe.
- Only the vocative singular form matches the original word
- 23 forms do not even contain the original word as a substring
- Changes occur in the very form of the word
- Redundancy inherent in the case inflections, rājabhyām

Duplicates occur in different cases and numbers for different types of words. There is large variety of word types, minor differences between the declension tables of seemingly similar words, huge number of exceptions to rules caused by individual words.

In the new work by the authors, computational algorithms for generatingsearch-related case-inflected forms of words, taking in to account all the above complexities involved in the problem has been developed.

The crux of the computational algorithm developed is as follows:

i. Apart from *Siddhānta-kaumudī*and *Kāśikā*, the book Śabdamañjarī[39], by Vidyasagar K. L. V. Sastry and Pandit L. AnantaramaSastri, was primarily used for *vibhakti*.This book contains the declension tables for Sanskrit nouns belonging to the various categories, and is acknowledged widely as a comprehensive consolidation of the relevant Pāṇinian rules.
ii. The input word is taken from the user along with the specification of its gender. The computational algorithm consists of two main steps, which are to identify the category of the given word and thereby the operations required to generate the required inflectional forms, and to then compute the inflectional forms by performing those operations.
iii. On studying the inflectional word forms thoroughly, it was found that the last part of a word is what changes when an inflectional form is produced, with only the last letter being affected in most cases. Based on this observation, a list of required basic operations on the last letter of words was identified.
iv. An exhaustive study and analysis of the declension tables specified as per Pāṇini's rules resulted in observations that led to the formation of the formulae for computing the inflectional forms. The formulae were simplified by the introduction of the pre-processing step.The introduction of functions that carry out some basic operations of euphonic conjunctions andthe pre-processing step has greatly enhanced the efficiency of the inflectional form generator.
v. A list of stems that are required to be appended to words in order to produce the inflectional forms, was identified for each category of words.
vi. A simple XML structure has been developed, which lists the operations required to compute the transformed words for each word category.
vii. Optimization of the computational model was carried out to facilitate fast search as explained in the search engine section of the document.

This efficiency of the algorithm over the existing approaches is as follows:

i. In the existing work, the rules for inflections are stored in tables, the tables are specific declensions by computing the *sandhi* of the stem and the suffix.
ii. As mentioned earlier the existing approach, applies *sup* rules of Pāṇini to the base word and sends the result to *sandhi* processing if any. Hence it requires *sandhi* processing for generating each of the 21 (7 * 3) case affixes. It also uses a database of nominal bases.

In a nutshell,

i. The current approach of the authors does not require access to database.
ii. Does not have the overhead of accessing the *sandhi* engine for generating each of the entry of the declension table.
iii. The formula has been presented in a simple XML structure whose schema once fixed need

not be changed. The XML structure by itself provides an algorithm description of the solution.

iv. The new work by the authors has identified and introduced basic operations and pre-processing steps for simplification of the solution. For example, the words pitṛ and dātṛ, though masculine and ending in the same vowel 'ṛ', produce different declension tables. The former gives rise to inflectional forms such as pitarau, pitaraḥ and pitaram, while the latter to forms such as dātārau, dātāraḥ and dātāram. However, before going in for computing the inflectional forms, the algorithm in this work generates forms, which are, respectively, pitar and dātār. Once this is done, there is no difference in the declension formulae for the two words.

v. Redundancies in generating the inflectional forms have been eliminated and an optimized algorithm developed for fast search.

vi. The algorithm sends the result of the declension engine to *sandhi* engine to see if further transformations occur due to *sandhi*, and generates those words and duly searches them in the target text.

vii. The algorithm has been thoroughly tested and hence is error free.

Hence the algorithm is simplified yet comprehensive and paves way for quick processing and there are only a few cases that need to be handled.

References

[1] Göttingen Register of Electronic Texts in Indian Languages- http://gretil.sub.uni-goettingen.de/

[2] Digital South Asia Library - http://dsal.uchicago.edu/search.html

[3] Digital Library of India - http://www.dli.ernet.in/

[4] Clay Sanskrit Library - http://www.claysanskritlibrary.org/corpus.php

[5] The Sanskrit Library - http://sanskritlibrary.org/

[6] Muktabodha Digital Library - http://muktalib5.org/digital_library_secure_entry.htm

[7] SARIT - http://sarit.indology.info/

[8] http://sanskrit.uohyd.ernet.in:8080/searchengine/index1.jsp http://sanskrit.uohyd.ernet.in:8080/searchengine/About.html http://sanskrit.uohyd.ernet.in/scl/morph/index.html

[9] Kasmir Raja S.V, Rajitha V., MeenakshiLakshmanan, "*Algorithms to Generate Case-inflected Forms of Feminine & Neuter Gender Nouns in Sanskrit for Fast and Slow Word Search"*

[10] Kasmir Raja S. V., Rajitha V., MeenakshiLakshmanan, "*Computational Algorithms Based on the Pāṇinian System to Process Euphonic Conjunctions for Word Searches"*, International Journal of Computer Science and Information Security, ISSN 1947-5500, Vol. 12, No. 8, 2014, pp 64-76.

[11] *Ravi Pal, "Advanced Heuristic Algorithm for Sandhi Processing in Sanskrit*", AKGEC Journal of Technology, Vol. 4, No.1, ISSN 0975-9514, Jan-June 2013.

[12] Gérard Huet, *"Towards Computational Processing of Sanskrit"*, ICON-2003, Proceedings Eds. Rajeev Sangal, S. M. Bendre and UdayaNarayana Singh, Central Institute of Indian Languages, Mysore, India, Dec. 2003., pp. 40-48.

[13] Gérard Huet, *"Automata Mista"*, In "Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday", Ed. NachumDershowitz, Springer-Verlag LNCS vol. 2772, pp. 359-372

[14] Gerard Huet, *"A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. J. Functional Programming"*, 15,4:573614, 2005.

[15] Gérard Huet, "*Lexicon-directed Segmentation and Tagging of Sanskrit*", XII[th] World Sanskrit Conference, Helsinki, Finland, Aug. 2003

[16] Gérard Huet, *"Transducers as lexicon morphisms, phonemic segmentation by euphony analysis, application to a sanskrit tagger"*, http://pauillac.inria.fr/~huet/FREE/tagger.pdf, 2002.

[17] Gérard Huet, "*Formal structure of Sanskrit text: Requirements analysis for a mechanical Sanskrit processor"*,Sanskrit Computational Linguistics, Lecture Notes in Computer Science, Volume 5402, 2009, pp 162-199.

[18] Malcolm D. Hyman, *"From PāṇinianSandhi to Finite State Calculus"*, Sanskrit Computational Linguistics, Lecture Notes in

Computer Science Volume 5402, 2009, pp 253-265

[19] Oliver Hellwig, *"SanskritTagger, a Stochastic Lexical and POS tagger for Sanskrit"*, Proceedings of First International Symposium on Sanskrit Computational Linguistics, Paris, October 29-31, 2007

[20] AmbaKulkarni& G Uma MaheshwarRao, *"Building Morphological Analysers and Generators for Indian Languages using FST"*, Tutorial at ICON 2010, IIT Kharagpur, Dec 2010

[21] AmbaKulkarni and DevanandShukl, *"Sanskrit Morphological Analyser: Some Issues"*, Festscrift volume of Bh. Krishnamoorty, Indian Linguistics, Vol 70, Nos 1-4, Dec 15, 2009

[22] Gerard Huet*,"Design of a Lexical Database for Sanskrit",* COLING Workshop on Electronic Dictionaries, Geneva, Aug. 29th, 2004, pp. 8-14.

[23] G. N. Jha, M. Agrawal, Subash, S. K. Mishra, D. Mani, D. Mishra, M. Bhadra and S. K. Singh, *"Inflectional Morphology Analyzer for Sanskrit"* A. Kulkarni, and G. Huet, editors, Sanskrit Computational Linguistics 1 and 2, pages 219-238, SpringerVerlag LNAI 5402, 2009.

[24] JhaGirishNath, October 2003, *"A Prolog Analyzer/Generator for Sanskrit Subanta Padas"*, Language in India, Volume 3, October 2003.
Full Paper - http://www.languageinindia.com/nov2003/jhasubanta.html
Related Paper – http://sanskrit.jnu.ac.in/subanta/Paper/CLTS.PDF
Working software - http://sanskrit.jnu.ac.in/subanta/generate.jsp

[25] Rajitha V., Kasmir Raja S.V., MeenakshiLakshmanan., "*An Ontology for Comprehensive Tutoring of Euphonic Conjunctions of Sanskrit Grammar"*, European Journal of Scientific Research, Vol. 124, No. 4, September 2014, pp. 460-467

[26] DīkṣitaBhaṭṭoji, *Siddhānta-kaumudī,* Translated by ŚrīśaCandraVasu, Volume 1, MotilalBanarsidas Publishers, Delhi, 1962.

[27] Vāmana, Jayāditya, *Kāśikā*, with the subcommentaries of Jinendrabuddhi, HaradattaMiśra and Dr. Jaya Shankar LalTripathi, Tara Book Agency, Varanasi, 1984.

[28] Kasmir Raja S. V, Rajitha V., MeenakshiLakshmanan, *"A Binary Schema and Computational Algorithms to Process Vowel-based Euphonic Conjunctions for Word Searches"*, International Journal of Applied Engineering Research, Volume 9, Number 20 (2014) pp. 7127-7142.

[29] Murray BarnsonEmeneau, *Sanskrit Sandhi and Exercises*, University of California Press, 1958

[30] Carol Whitfield, *Handbook for The Online Sanskrit Audio Course Part Two: Sandhi Rules*, http://www.arshabodha.org.

[31] *Sanskrit Tutorial*, http://www.learnsanskrit.org/references/*sandhi*

[32] *Sanskrit Lessons*, http://www.chitrapurmath.net/sanskrit/

[33] Vyoma Linguistics Labs Foundation, http://vyomalabs.in/

[34] Gerard Heut, *Sandhi Engine*, http://sanskrit.inria.fr/DICO/*sandhi*.en.html

[35] Sanskrit *Sandhi* tool, http://www.greenmesg.org/sanskrit_online_tools/sanskrit_*sandhi*_tool.php

[36] Prof. G.S.S. Murthy, *Sandhi Joiner program*, http://www.iiitb.ac.in/Chitrakavya/learnsanskrit.html

[37] Sanskrit *Sandhi* Generator, Special center for Sanskrit Studies, Jawaharlal Nehru University, http://sanskrit.jnu.ac.in/*sandhi*/gen.jsp

[38] Kasmir Raja S V, Rajitha V, MeenakshiLakshmanan, *"Computational Model to Generate Case-Inflected Forms of Masculine Nouns for Word Search in Sanskrit E-Text"* Journal of Computer Science 12/2014; 10(11):2260-2268.

[39] Vidyasagar K. L. V. Sastry and Pandit L. AnantaramaSastri, *Śabdamañjarī*, R. S. Vadhyar& Sons Publishers, Palghat, India, revised edition 2002.

[40] SmitaSelot, A.S. Zadgaonkar And Neeta Tripathi, *"Subantapada analyzer for Sanskrit",* Oriental Journal of Computer Science & Technology, Vol. 3(1), 89-93 (2010)

[41] Academy of Sanskrit Research, Melkote, Karnataka. http://www.sanskritacademy.org/