International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-4, November 2019

Computational Algorithms for Deep and Shallow Word Search in Sanskrit with Case-inflected Forms of Feminine and Neuter Nouns

Rajitha V., Suganya R., Meenakshi Lakshmanan

Abstract*: *Effective and comprehensive word-search in Sanskrit E-text is a non-trivial problem in natural language processing that has been solved in this work for the first time in the literature. The problem is also one of great benefit to a variety of users. The complexity of the problem stems from the fact that words in Sanskrit can get completely metamorphosed on account of the phenomena of euphonic conjunctions and case-inflected forms. This work deals with the case-inflected forms of feminine and neuter gender nouns in particular, and their bearing on the search problem. The novel computational model presented here for generating case-inflected forms of Sanskrit nouns involves a categorization of word forms and a development of new formulae to achieve the generation of case-inflected forms. Pre-processing to improve the performance of the algorithm based on these formulae has also been developed. This model has been extended to include processing of euphonic conjunctions to enable a more elaborate search and to thereby produce meaningful search results. The dual paradigms of deep/slow and shallow/fast searching have been handled in this work and the efficiency of the fast search algorithm developed is shown to have an increased efficiency of 77-80%. The work also discusses the trade-off between search accuracy and speed in the context of word-search in Sanskrit E-text.

Keywords*: *case inflections, vibhakti, word search, Sanskrit, declension

I. INTRODUCTION

The discovery, by scientists at NASA [1] and other acclaimed researchers, that Sanskrit is the most scientific natural language and the one most suited for computing, has won universal recognition for the language. There has been an increased global recognition of the importance of the Sanskrit language amongst scientists especially from the last couple of decades of the twentieth century. In fact, it has been recognized that the structuring format of the grammar adopted by the fifth century grammarian Pāṇini, [2][3] is what twentieth century researchers Backus and Naur developed as a specification for formal languages [4][5] and some have even chosen to rename the specification as the 'Pāṇini-Backus Form' [6][7]. The language's object-oriented approach, the high degree of rigor ensconced in it and the perfectly phonetic nature of the language are factors that have contributed to its winning such international recognition among the computing fraternity. Further, natural language processing in Sanskrit [8] can pay rich dividends, for it is acknowledged as the mother of most Indian languages and is also closely related to some Eurasian languages. Hence, a solution to a problem in Sanskrit could provide solution supersets to problems of related or derived languages.

The importance of Sanskrit stems not only from the above considerations, but also from the fact that it has a huge repertoire of knowledge ranging from *Āyurveda*, *Yoga*, the Fine Arts and Politics to Engineering, Science, Metaphysics and Spirituality. The stumbling block in accessing this colossal cache of knowledge, however, is that the Sanskrit language by itself constitutes a highly complex and evolved system of communication [9]. It is precisely formulated at lower levels such as the morphological and syntactic, but within the limits of complex structures and rules, creates an explosion of possibilities at higher levels such as the semantic. Hence, new compound words can be created in the language, making it the perfect medium for poets. Furthermore, it is not a widely spoken or used language anymore, though efforts are on to revive it, which makes information about it that much harder to filter out and use.

Nevertheless, owing to the widespread interest in the Sanskrit language as well as its literature not only in India, the land of its birth, but also globally, and with the increased availability of large repositories of Sanskrit works in the digital format [10][11], it is becoming more and more important to provide comprehensive natural language processing solutions in Sanskrit, for the benefit of a variety of current and potential users.

A. The Word Search Problem in Sanskrit

Word search [12][13] is a rather fundamental and perhaps the most widely utilized language-level application for any patois. In the context of Sanskrit, different categories of users may wish to search for words or phrases in Sanskrit works. For example, a researcher in History may want to know the various instances of a particular word or phrase in a text, to analyze the period of the work or to determine its author, the author's affiliations, etc.

Manuscript published on November 30, 2019.
* Correspondence Author
Rajitha V., Department of Computer Science, Meenakshi College for Women, Chennai, India. Email: warftrajitha@gmail.com
Suganya R.*, Department of Computer Science, Meenakshi College for Women, Chennai, India. Email: suganyaramadoss@gmail.com
Meenakshi Lakshmanan, Department of Computer Science, Meenakshi College for Women, Chennai, India. Email: meenakshi.lakshmanan@gmail. com

An artist may wish to access all information regarding a particular *mudrā* or posture in ancient treatises on the classical dance form, *Bharatanaṭyam*. A scientist may want to access all references to a particular herb in a text on *Āyurveda*, the ancient system of medicine.

The presence or absence of certain phrases and the frequency of occurrence of specific words have been proposed and used as factors to investigate the authorship of several philosophical works. In a couple of instances, certitude about the position of the author on an issue of importance has been arrived at by examining whether the author has, anywhere in his/her work, used two key words synonymously. Pundits are normally in a position to recall many words and phrases of the traditional, authoritative texts that they expound and often specify portions of texts by means of key words and phrases. Several scholarly debates and books have focused on where and how some profound words have been used in scriptural texts.

People who wish to search for words or phrases in printed Sanskrit books find it very difficult to do so because there is no extensive table of contents or detailed word indices in most of these books. Hence manual search becomes inevitable. It can well be imagined how tedious and time consuming this would be specially when there are hundreds of thousands of verses in a single text, and also considering the hundreds of thousands of works available in Sanskrit. However efforts of various agencies across the globe are fructifying and Sanskrit works are increasingly being made available for reference as E-Texts in digital repositories.

In view of this availability for reference of a large repertoire of Sanskrit E-texts and the importance and usefulness of automated word-searches for a wide variety of users, it is the need of the hour to develop a computational algorithm for comprehensively and effectively locating specified words and phrases in Sanskrit E-texts. Given that the Unicode standard is used, it may be expected that searching for words or phrases in Sanskrit E-texts would require simple text pattern matching techniques already available in word processors or document viewers. However, it is demonstrably insufficient to search for just the specified word as is done in the case of English and as is generally being done at present even in the case of Sanskrit E-texts. This is because the ubiquitous phenomena of euphonic conjunctions and case-inflected forms in Sanskrit cause dramatic changes in the basic form of a word, unlike in other languages.

For example, the search word "*gurubhyaḥ*" (meaning 'to/for/from/than the teachers') might be found very often in the text as *gurubhyas* or *gurubhyo* because of the operation of euphonic conjunction (*sandhi*) rules caused by adjacent words. Similarly, the search word *asamardhiḥ* might be found in the text as *āsamardhiḥ*, *āsamardhis*, *āsamardhir*, *asamarddhiḥ* or *asamardddhiḥ*. *Sandhi* can thus cause internal or external transformations in a word.

Likewise, a noun that is of interest may be encountered in various cases or '*vibhaktis*', viz. the nominative, accusative, instrumental, dative, ablative, genitive, locative or vocative case. Further, the noun could be used in the singular, dual or plural. For instance, the basic noun '*śvan*' (meaning 'dog') gets transformed into the form '*śune*' when used in the dative singular (meaning 'to / for the dog') and into the form '*śunām*' when it is used in the possessive plural (meaning 'of the dogs'). Decidedly, a search merely for the word '*śvan*' would not yield complete results. Further, a search for '*śunām*', for instance, would identify only one of the many case-inflected forms of the word.

A simple example that illustrates the importance of factoring in the aspects of euphonic conjunctions and case-inflected forms for word search in Sanskrit, is presented below. A search for the word '*buddhiḥ*' (nominative singular form) meaning 'intellect', in the Sanskrit philosophical work, *Bhagavadgītā*, could be required by, for instance, a philosophy student or teacher who wishes to investigate the contexts in which the word has been used in the *Bhagavadgītā*, to determine various connotations of the word as used in the text. Given in the Table-I is a list of possible word forms of the given word, created due to the influence of euphonic conjunctions or case-inflections.

It must be mentioned here that Table-I is far from an exhaustive list of the various forms of the word '*buddhi*' that can be generated, and is only meant to be illustrative. Forms that are generated through euphonic conjunctions or case-inflections but not found in the *Bhagavadgītā* are not listed in the table.

Table - I: Number of occurrences of the forms of the word '*buddhi*' (intellect) in the *Bhagavadgīta*

#	Search word	Euphonic Conjunction (E) / Case-inflection (V)	Details of case-inflection	Number of occurrences
1	*buddhi*	-	-	41
2	*buddhiḥ*	V	Nominative singular	6
3	*buddhis*	E	-	2
4	*buddhir*	E	-	17
5	*buddhim*	V	Accusative singular	3
6	*buddhiṛ*	E	-	2
7	*buddhin*	E	-	2
8	*buddheḥ*	V	Ablative, Genitive singular	2
9	*buddher*	E	-	2
10	*buddhī*	V	Nominative, Accusative, Vocative dual	0
11	*buddhe*	E	-	4
12	*buddhay*	E	-	5
13	*buddhau*	V	Locative singular	1
14	*buddhayaḥ*	V	Nominative, Vocative plural	2
15	*buddhayas*	E	-	1
16	*buddhyā*	V	Instrumental singular	4

The *Bhagavadgītā* is a 700-verse long poem, consisting of more than 9,000 words. It must also be stated here, that this text is a rather small treatise when compared to other famous Sanskrit texts like the *Rāmāyaṇa* (24,000 verses) and *Mahābhārata* (100,000 verses).

Now if the word searched for is '*buddhiḥ*', the nominative singular form, which is the form of the word when used as the subject of a sentence, then we would be able to locate only 6 occurrences in the text.

However, the search words shown in rows 3 and 4 are alternative forms of the word '*buddhiḥ*', caused by the play of euphonic conjunctions. Hence, searching for '*buddhiḥ*' does not return as many as 19 occurrences of the word, because, the word is present in a slightly altered form in the text.

Similarly, searching for '*buddhim*', the accusative singular form (as the object of a sentence), yields two instances, but omits 4 (rows 6 and 7).

Further, if the search is for the root word '*buddhi*' itself, then all of the forms listed in rows 2-7 are also subsumed in the search, whereby the number of occurrences 41 of '*buddhi*' includes the total number of occurrences 32 of the words in rows 2-7. (The remaining 9 word forms include those such as '*buddhimān*' meaning 'one who is endowed with an intellect', which are not generated from euphonic conjunctions or case-inflections). Hence, using the root word may seem to be a better option. However, the rest of the table clearly shows that important occurrences of the words in forms such as '*buddheḥ*' meaning 'of the intellect', or '*buddhyā*' meaning 'by/with/through the intellect', would not be detected in the search. In all, with the above restricted table itself, there are 14 independent instances (after eliminating the overlaps) that will not be recognized.

This example clearly illustrates that there may be crucial occurrences of the word that may be omitted in a Sanskrit word search if the dual phenomena of euphonic conjunctions and case-inflections are not factored into the search.

II. THE UNIQUENESS OF THIS WORK

The problem of comprehensively searching for words or phrases in Sanskrit text has hence posed a challenge, and no comprehensive solution to it has been devised in the literature [14][15][16][17][18]. This and what has been detailed in the previous section set up the case for the need to develop a comprehensive solution for the word search problem in Sanskrit through the efficient generation of all the required case-inflected forms of the word of interest and the possible forms of the word that could arise as a result of euphonic conjunctions.

Nouns belonging to any of the three genders in Sanskrit (masculine, feminine and neuter) are dealt with in the categories of ordinary words (*sādhāraṇa-śabda*) and special words (*viśeṣa-śabda*). These are further grouped into vowel-ending and consonant-ending words. The *sādhāraṇa-śabdas* of feminine and neuter genders of both the vowel-ending and consonant-ending types are dealt with in this work. A computational model has been developed to generate all possible case-inflected forms of such words for the purpose of word search. Aspects of the model for masculine gender nouns have been published [19]. Also published is a novel and comprehensive computational model that generates all possible forms of words in accordance with the rules of internal and external euphonic conjunctions [20].

What is presented herein is an optimized computational model developed vis-à-vis the feminine and neuter gender words. Using this case-inflected model developed here along with the one developed earlier for masculine gender words [19] and the euphonic conjunctions model developed earlier [19][20], this work also presents a comprehensive and effective solution to the complex problem of word search in Sanskrit e-Text. Finally, this work presents an analysis of both the computational models in terms of search accuracy and speed.

III. AN OVERVIEW OF THE PROPOSED SOLUTION TO THE PROBLEM OF GENERATING FORMS OF WORDS GENERATED BY EUPHONIC CONJUNCTIONS

A *sandhi* is a point in a word or between words, at which adjacent letters coalesce and transform. This is a common feature in many Indian languages as against European languages, and has far-reaching consequences in Sanskrit. The transformation caused by the application of rules of *sandhi* in Sanskrit can be significant enough to alter the word itself to such a degree that the transformed word would not show up in a simple word search.

IV. THE COMPLEXITIES INHERENT IN GENERATING CASE-INFLECTED FORMS OF WORDS

As mentioned, in Sanskrit, declensions of nouns are based on eight cases and three numbers – singular, dual and plural. This gives rise to 24 case-inflected forms for each noun, whichever of the three genders the noun may belong to. It must be noted here that a word itself, and not just the object it denotes, has a gender in Sanskrit. For example, both the words '*vidyālayaḥ*' and '*pāṭhaśālā*' mean the same thing – school - though the first word is masculine, and the second, feminine. A sample declension for the feminine word 'मति' ('*mati*') also meaning 'intellect', is given in Table-II. Clearly, the word '*mati*' is so totally changed in its various case-inflections, and hence most of the forms in which it could be found in a text (such as in the common form 'in the intellect', which is '*matyām*' or '*matau*'), would go unnoticed if a simple search for '*mati*' is done. Also, there could be duplicates within the 24 inflected forms of the word, which need to be eliminated in the context of word-search. It must be stated here that the duplicates occur in different cases and numbers for different types of words, and hence this aspect is also not uniform in Sanskrit. Furthermore, words could have more than one form for a particular case and number. For instance, in Table-II it can be seen that the dative, ablative, possessive and vocative singular of the word '*mati*' have more than one form. This again is not uniform and the case and number in which more than one form is encountered, differs across words.

Table-II: Case-inflections of the *i*-ending feminine word *mati* (meaning 'intellect') (Standard Sanskrit declension table)

#	Case	Singular	Dual	Plural
1	Nominative (subject)	मतिः *matiḥ*	मती *matī*	मतयः *matayaḥ*
2	Accusative (object)	मतिं *matiṁ*	मती *matī*	मतीः *matīḥ*
3	Instrumental (by, with, through)	मत्या *matyā*	मतिभ्याम् *matibhyām*	मतिभिः *matibhiḥ*
4	Dative (for, to)	मत्यै , मतये *matyai, mataye*	मतिभ्याम् *matibhyām*	मतिभ्यः *matibhyaḥ*

5	Ablative (from, than)	मत्याः , मतेः *matyāḥ, mateḥ*	मतिभ्याम् *matibhyām*	मतिभ्यः *matibhyaḥ*
6	Possessive (belongs to, has/have)	मत्याः , मतेः *matyāḥ, mateḥ*	मत्योः *matyoḥ*	मतीनाम् *matīnām*
7	Locative (in, on, at)	मत्याम् , मतौ *matyām, matau*	मत्योः *matyoḥ*	मतिषु *matiṣu*
8	Vocative (calling out)	मते *mate*	मती *matī*	मतयः *matayaḥ*

V. THE GRAMMATICAL BASIS

The *Aṣṭādhyāyī* (work in eight chapters) [21][22][23] is a renowned text authored by the ancient grammarian Pāṇini, and is universally acknowledged as the final authority on Sanskrit grammar. The work has almost 4000 tersely stated aphorisms arranged in a precise order, that lay out the grammatical rules of Sanskrit at the morphological, phonological, syntactic and semantic levels, and sometimes even at the level of discourse. The work enunciates almost 400 complex aphorisms that contain rules to generate the case-inflected forms of nouns, as categorized in the *Siddhānta-kaumudī* [24], an authoritative commentary on the original. Another commentary revered by scholars is the *Kāśika* [25][26]. The rules laid down in the succinct aphorisms of Pāṇini are used to build the declension tables (set of all case-inflected forms) of any given noun. The difficulty in writing an efficient computer-executable program to generate these tables springs from the large variety of words and word types, minor differences between the declension tables of seemingly similar words, and the huge number of exceptions to rules caused by individual words. Moreover, the reason for not using Pāṇini's rules directly for this solution is because they involve complex, sequential rule processing that would introduce latency in the primary solution for the search problem.

The book *Śabdamañjarī* [27] enumerates the declension tables for Sanskrit nouns belonging to the various categories, and is used as a handy and comprehensive text book on case-inflected forms in Sanskrit. The book consolidates the relevant Pāṇinian rules into the final product, viz. the declension tables for various words or word groups. This text has been used as a primary basis for this work, with the book *Kāśika* [25][26] used to derive further information whenever required.

A. The Sanskrit Alphabet

The Sanskrit alphabet consists of 48 basic letters, with 13 vowels (both long and short), 4 semi-vowels, 29 consonants, and two special letters, viz. the *anusvāra* and the *visarga*. The written script for Sanskrit goes by the name '*Devanāgarī'*. It is an accepted norm now for E-texts to be in either *Devanāgarī* Unicode, or in the Latin Unicode, with special diacritical letters provided for the various letters of the Sanskrit alphabet. This work makes use of the Latin Unicode for processing, though examples are provided using the *Devanāgarī* script as well. Sanskrit words in Latin Unicode are indicated in italics as per the prevalent norm.

B. Definitions Developed in this Work

Before going into the specification of the formulae developed in this work to compute the inflected forms, it is necessary to introduce some terminology developed exclusively by the authors.

For any vowel x, the list of operations defined on x are listed in Table-III. The first two of these operations is respectively the short and long forms of vowels, and the latter four are *sandhi*-based transformations.

The consonants of Sanskrit consist of mutes, sibilants and the aspirate. The mutes are given in Table-IV which has the rows and columns conveniently numbered.

In Table-IV, letters of Columns 2 and 4 appear with two letters each, and are the aspirate forms of the corresponding letters in Columns 1 and 3 respectively. For instance the letter represented here as '*kh*' is actually the single letter 'ख्' in the *Devanāgarī* script. All letters in Columns 1 and 2 are hard consonants and those in Columns 3 and 4 are soft consonants. Column 5 comprises the nasal consonants. The sibilants are '*ś*', '*ṣ*' and '*s*' and the aspirate is '*h*'. The special characters *anusvāra* and *visarga* are represented as '*ṁ*' and '*ḥ*' respectively.

On studying the inflected word forms in detail, it was determined that the last portion of a word is what changes when an inflected form is produced, with only the last letter being affected in a majority of cases. Based on this observation, a list of required basic operations on the last letter of words was identified.

Let X be the given search word and let x denote its last letter. If x is a vowel, let x_d denote the *dīrgha* (lengthened form) of x, x_r the *hrasva* (shortened form) of x, x_g the *guṇa* of x, x_v the *vṛddhi* of x, x_a the *ayāyāvāva* equivalent of x, and x_y the *yaṇ* equivalent of x as defined in Table-III. If x is a mute consonant, represented as x_{ij} in Table-IV, let x_s denote soften$(x) = x_{i3}$, x_h harden$(x) = x_{i1}$, x_n nasalize$(x) = x_{i5}$ and x_i rowShift$(x_{ij}, k) = x_{kj}$, $i \in \{1,2,3,4,5\}$. For any letter x, let x_l denote the *lopa* or deletion of the letter x from the word X.

The operations denoted by the suffixes can be performed in succession and appropriately denoted. For instance, x_{ga} indicates that the *guṇa* operation is first applied to x and then the *ayāyāvāva* operation is applied to the resultant. In an operation involving *lopa*, such as x_{ld} the deletion of x is done and then operation denoted by the suffix d is applied to the new last letter of X. The operand '+' between letters or groups of letters denotes string concatenation. String literals are denoted within double quotation marks during concatenation.

A list of stems δ_i that are required to be appended to words in order to produce the inflected forms, was identified for each category of words. Table -V lists the stems identified. The column δ denotes δ_i and the column 'Stem' gives the value of δ_i.

Clearly, some of the stems in this list can be constructed by appending two or more other stems, but such redundant compound stems were not eliminated from the list because they aid in simpler processing.

Table - III: Operations on vowels

#	*x*	*dīrgha*	*hrasva*	*guṇa*	*vṛddhi*	*ayāyāvāva*	*yaṇ*
1	*a*	*ā*	*a*	*a*	*ā*	-	-
2	*ā*	*ā*	*a*	-	-	-	-
3	*i*	*ī*	*i*	*e*	*ai*	-	*y*

4	ī	ī	i	e	ai	-	Y
5	u	ū	u	o	au	-	v
6	ū	ū	u	o	au	-	v
7	ṛ	ṝ	ṛ	ar	ār	-	r
8	ṝ	ṝ	ṛ	ar	ār	-	r

9	ḷ	ḝ	ḷ	al	āl	-	l
10	e	e	-	e	ai	ay	-
11	o	o	-	o	au	av	-
12	ai	ai	-	-	-	āy	-
13	au	au	-	-	-	āv	-

Table - IV: Mute consonants

#	1	2	3	4	5
1	k	kh	g	gh	ṅ
2	c	ch	j	jh	ñ
3	ṭ	ṭh	ḍ	ḍh	ṇ
4	t	th	d	dh	n
5	p	ph	b	bh	m

Table - V: Stems δ_i used in creating inflected forms

δ	Stem	δ	Stem	δ	Stem	δ	Stem	δ	Stem
1	ḥ	15	uḥ	29	ai	43	ai	57	ñci
2	au	16	bhyaḥ	30	y	44	ye	58	ī
3	aḥ	17	t	31	r	45	vaḥ	59	ne
4	āḥ	18	yoḥ	32	u	46	auḥ	60	ani
5	am	19	oḥ	33	naḥ	47	āni	61	aye
6	m	20	ām	34	nau	48	noḥ	62	ū
7	n	21	sya	35	ān	49	nti	63	ave
8	a	22	i	36	nam	50	di	64	voḥ
9	ā	23	nām	37	āṁs	51	ni	65	śu
10	bhyām	24	su	38	yā	52	nī	66	d
11	bhiḥ	25	ṣu	39	yai	53	ṁśi	67	p
12	aiḥ	26	āy	40	yāḥ	54	ṁsi	68	j
13	e	27	ena	41	yām	55	ṁṣi	69	ntī
14	āya	28	nā	42	yaḥ	56	ṁhi	70	o
71	s								

VI. THE COMPUTATIONAL ALGORITHM TO GENERATE CASE-INFLECTED FORMS

The control abstraction shown in Algorithm 1 encapsulates the steps involved in generating the search-related case-inflections, and was evolved during work on the declension of masculine nouns [19]. The input word X is taken from the user along with the specification of its gender g; x is the last letter of X. The steps of this control abstraction are detailed in the sections that follow.

ALGORITHM 1. GenerateInflections($\boldsymbol{X}$, $\boldsymbol{g}$)

// X is the given word and g its gender;
// Let x be the last letter of X;
Step 1: Find the word category C (if any), using g and x;
Step 2: Compute x', the basic transformation of x based on g and C;
Step 3: Parse the XML schema to retrieve the formulae for this combination of x, g, C;
Step 4: Perform operations specified in the formulae to generate the inflected forms of $\boldsymbol{X}$;

VII. STEP1: WORD CATEGORIZATION

The first step of the control abstraction is to determine the category of the word. Tables-VI and VII were developed after a thorough study of the declension tables of the various categories of feminine and neuter words respectively. These tables provide the bases for the development of methods to complete the computations specified in the first two steps of the algorithm specified in Algorithm 1.

As seen from these tables, the category of a word is a numerical value that determines the sub-species of the word within a particular gender and last-letter. For instance, there are four species of neuter gender words ending in '*n*'. The category assumes importance because it too determines the declension formulae for the given word.

Barring a few exceptions, the only way to determine the category of a noun as indicated by Pāṇini and also subsequent commentaries on his work, is by delineation. For example, given the feminine gender word, '*latā*' (meaning 'creeper'), determining its category (whether it belongs to Category 1 like *sītā* or to Category 2 like *ambā* of the *ā*-ending feminines) is not computable, but is only enumerable. The category of a given word is determined using the method of hashing, and in particular, using hash indices.

Table - VI: Computation of x'/X' for feminine gender nouns

#	x	Category	x'	Example		Operation to compute x'
				X	X'	
1	ā	1	e	sītā	sīte	x_l + "e"
2		2	e	ambā	ambe	x_l + "e"
3	i	-	y	mati	maty	x_y
4	ī	1	y	nadī	nady	x_y
5		2	y	śrī	śriy	x_r + "y"
6		3	y	strī	striy	x_r + "y"
7	u	-	v	dhenu	dhenv	x_y
8	ū	1	v	vadhū	vadhv	x_y
9		2	v	bhū	bhuv	x_r + "v"
10	ṛ	1	ṛ	svasṛ	svasār	x_v
11		2	ṛ	mātṛ	mātar	x_g
12	o	-	au	go	gau	x_v
13	ai	-	y	rai	rāy	x_a
14	au	-	v	nau	nāv	x_a
15	c	-	j	vāc	vāj	x_s
16	j	-	k	sraj	srak	x_{h1}
17	t	-	d	sarit	sarid	x_s
18	d	-	t	śarad	śarat	x_h
19	dh	-	t	kṣudh	kṣut	x_h
20	n	-	n	sīman	sīmn	x_{ll} + "n"
21	p	-	d	ap	ad	x_{4s}
22	bh	-	p	kakubh	kakup	x_h
23	r	-	-	gir	gī	x_{ld}
24	v	-	y	div	dy	x_{ly}
25	s	1	-	bhās	bhā	x_l
26		2	ṣ	āśis	āśiṣ	x_3
27	h	-	t	upānah	upānat	x_l + "t"

Table VII: Computation of x'/X' for neuter gender nouns

#	x	Category	x'	Example		Operation to compute x'
				X	X'	
1	a	-	-	phala	phal	x_l
2	i	1	ī	vāri	vārī	x_d
3		2	n	dadhi	dadhn	x_l + "n"
4		3	ī	śuci	śucī	x_d
5	u	1	ū	madhu	madhū	x_d
6		2	ū	guru	gurū	x_d
7	ṛ	-	ṝ	dātṛ	dātṝ	x_d
8	c	-	k	suvāc	suvāk	x_1
9	j	-	k	asṛj	asṛk	x_{h1}
10	t	1	d	jagat	jagad	x_s
11		2	d	dadat	dadad	x_s
12		3	d	tudat	tudad	x_s
13		4	d	pacat	pacad	x_s
14		5	d	mahat	mahad	x_s
15	d	-	t	hṛd	hṛt	x_h

#	x	Category	x'	Example		Operation to compute x'
				X	X'	
16	n	1	n	*nāman*	*nāmn*	x_{ll} + "n"
17		2	-	*karman*	*karma*	x_l
18		3	n	*ahan*	*ahn*	x_{ll} + "n"
19		4	-	*guṇin*	*guṇi*	x_l
20	r	-	-	*vār*	*vā*	x_l
21	*ś*	-	*k*	*tādṛś*	*tādṛk*	x_l + "k"
22	*ṣ*	-	*ṭ*	*sutviṣ*	*sutviṭ*	x_l + "ṭ"
23	*s*	1	*o*	*manas*	*mano*	x_{ll} + "o"
24		2	*ṣ*	*havis*	*haviṣ*	x_3
25		3	*ṣ*	*vapus*	*vapuṣ*	x_3
26		4	*ṣ*	*tasthivas*	*tasthuṣ*	x_{lll} + "uṣ"
27	*h*	-	*ṭ*	*ambhoruh*	*ambhoruṭ*	x_l + "ṭ"

VIII. STEP 2: THE PRE-PROCESSING STEP OF COMPUTING X'

X' is obtained by transforming the given word X through operations performed on x; x' denotes the last letter of X'.

Analysis of the declension tables of the various types and categories of nouns, led to the derivation of the value of x'/X' from x/X, depending on g, x and the category C.

Table-VI encapsulates the derivation of X' for the feminine word categories and Table-VII gives similar information for neuter gender words. Sample words for each category too are provided in the said tables.

The pre-processing step of computing X', has significantly enhanced the efficiency of the algorithm. This can be illustrated using an example. Consider the two categories of *ṛ*-ending feminines as shown in Table-VI. Sample words for the categories are '*svasṛ*' and '*mātṛ*' respectively. Some of the forms of '*svasṛ*' are '*svasāram*', '*svasārau*' and '*svasāraḥ*', while the corresponding forms for '*mātṛ*' are '*mātarau*', '*mātaram*' and '*mātaraḥ*'. The rest of the declension table is the same for both the words. As can be seen in Tabl-VI, X' for these words have been identified as '*svasār*' and '*mātar*' respectively, both of which have been got by appropriate transformations on the common last letter x of the words. Once X' has been thus computed, it is found that the formulae for all the 24 inflected forms of these words, are identical.

IX. STEP 3 AND 4: FORMULAE TO COMPUTE THE INFLECTIONAL FORMS

An XML structure has been developed to extract formulae for the inflected forms for every single category of words.

The XML file is parsed to retrieve the formulae for a given word (specifically, its last letter), gender and category, and the simple computations indicated by the formulae are carried out to generate the various inflected forms. The XML structure developed for feminine words ending in '*ā*' and those ending in *'i'* are given in Algorithm 2. XMLStructureFeminine.

ALGORITHM 2. XMLStructureFeminine

```
<Gender G = "feminine">
  <LastLetter L = "ā">
    x, x', x + δ1, x + δ6, x_r + δ38, x + δ10, x + δ11, x + δ39, x + δ16, x + δ40, x_r + δ18, x + δ23, x + δ41, x + δ24
    <Category C = "2">
      x_r
    </Category>
  </LastLetter>
  <LastLetter L = "i">
    x + δ1, x_d, x_ga + δ3, x + δ6, x_d + δ1, x' + δ9, x + δ10, x + δ11, x' + δ43, x_ga + δ13, x + δ16, x' + δ4, x_g + δ1, x' + δ19, x_d + δ23, x' + δ20, x_l + δ2, x + δ25, x_g
  </LastLetter>
</Gender>
```

The XML structure has been optimized to contain common formulae in the upper levels of the hierarchy, to allow for quicker processing. The formulae that denote the transformations are specified as a comma-separated list in the structure. A unique feature of this XML structure is that it represents an algorithm by itself apart from acting as a hierarchical organization of data.

The inflected forms obtained as a result of the operations in Algorithm 2 are illustrated in Table-VIII, which also shows how redundancies in the inflected forms of various cases have been eliminated within the XML structure itself.

Similarly, the XML structure for sample words of the neuter gender ending in the letters '*i*' and '*c*' is given in Algorithm 3.

The inflected forms computed for sample words of this gender are shown in Table-IX.

It must be mentioned here that the word '*vārinī*', the nominative dual of '*vāri*' as seen in Table-IX, will further get transformed into '*vāriṇī*' due to a sandhi rule that will operate at a later stage through the Sandhi Engine as detailed in Section XI below.

ALGORITHM 3. XMLStructureNeuter

```
<Gender G = "neuter">
  <LastLetter L = "i">
    x, x + δ52, x_g, x + δ10, x + δ11, x + δ16, x + δ25
    <Category C = "1 OR 2">
      x' + δ51, x + δ28, x + δ59, x + δ33, x + δ48, x' + δ23, x + δ51
      <Category C = "2">
        x_l + δ61, x_g + δ1, x_y + δ19, x_l + δ2
      </Category>
    </Category>
    <Category C = "3">
      x_d + δ51, x' + δ9, x' + δ13, x' + δ3, x' + δ19, x' + δ20, x' + δ22, x_l + δ60
    </Category>
  </LastLetter>
  <LastLetter L = "c">
    x', x + δ58, x_l + δ57, x + δ9, x'_s + δ10, x'_s + δ11, x + δ13, x'_s + δ16, x + δ3, x + δ19, x + δ20, x + δ22, x' + δ25
  </LastLetter>
</Gender>
```

Table -VIII: Inflected forms computed for sample feminine words

X	Vibhakti: number	Formula	Inflected form
sītā	1: singular	x	*sītā*
	1: dual, 2: dual, 8: singular, 8: dual	x'	*site*
	1: plural, 2: plural, 8: plural	$x + \delta_1$	*sītāḥ*
	2: singular	$x + \delta_6$	*sītām*
	3: singular	$x_r + \delta_{38}$	*sītayā*
	3: dual, 4: dual, 5: dual	$x + \delta_{10}$	*sītābhyām*
	3: plural	$x + \delta_{11}$	*sītābhiḥ*
	4: singular	$x + \delta_{39}$	*sītāyai*
	4: plural, 5: plural	$x + \delta_{16}$	*sītābhyaḥ*
	5: singular, 6: singular	$x + \delta_{40}$	*sītāyāḥ*

	6: dual, 7: dual	$x_r + \delta_{18}$	*sītayoḥ*
	6: plural	$x + \delta_{23}$	*sītānām*
	7: singular	$x + \delta_{41}$	*sītāyām*
	7: plural	$x + \delta_{24}$	*sītāsu*
ambā	1: singular	x	*ambā*
	1: dual, 2: dual, 8: dual	x'	*ambe*
	1: plural, 2: plural, 8: plural	$x + \delta_1$	*ambāḥ*
	2: singular	$x + \delta_6$	*ambām*
	3: singular	$x_r + \delta_{38}$	*ambayā*

	3: dual, 4: dual,5: dual	$x + \delta_{10}$	*ambābhyām*
	3: plural	$x + \delta_{11}$	*ambābhiḥ*
	4: singular	$x + \delta_{39}$	*ambāyai*
	4: plural, 5: plural	$x + \delta_{16}$	*ambābhyaḥ*
	5: singular, 6: singular	$x + \delta_{40}$	*ambāyāḥ*

	6: dual, 7: dual	$x_r + \delta_{18}$	*ambayoḥ*
	6: plural	$x + \delta_{23}$	*ambānām*
	7: singular	$x + \delta_{41}$	*ambāyām*
	7: plural	$x + \delta_{24}$	*ambāsu*
	8: singular	x_r	*amba*
mati	1: singular	$x + \delta_1$	*matiḥ*
	1: dual, 2: dual, 8: dual	x_d	*matī*
	1: plural, 8: plural	$x_{ga} + \delta_3$	*matayaḥ*
	2: singular	$x + \delta_6$	*matim*
	2: plural	$x_d + \delta_1$	*matīḥ*
	3: singular	$x' + \delta_9$	*matyā*
	3: dual, 4: dual, 5: dual	$x + \delta_{10}$	*matibhyām*
	3: plural	$x + \delta_{11}$	*matibhiḥ*
	4: singular	$x' + \delta_{43}$	*matyai*
	4: singular	$x_{ga} + \delta_{13}$	*mataye*
	4: plural, 5: plural	$x + \delta_{16}$	*matibhyaḥ*
	5: singular, 6: singular	$x' + \delta_4$	*matyāḥ*
	5: singular, 6: singular	$x_g + \delta_1$	*mateḥ*
	6: dual, 7: dual	$x' + \delta_{19}$	*matyoḥ*
	6: plural	$x_d + \delta_{23}$	*matīnām*
	7: singular	$x' + \delta_{20}$	*matyām*
	7: singular	$x_l + \delta_2$	*matau*
	7: plural	$x + \delta_{25}$	*matiṣu*
	8: singular	x_g	*mate*

Table-IX: Inflected forms computed for sample neuter words

X	Vibhakti: number	Formula	Inflected forms
vāri	1: singular, 2: singular, 8: singular	x	*vāri*
	1: dual, 2: dual, 8: dual	$x + \delta_{52}$	*vārinī*
	1: plural, 2: plural, 8: plural	$x' + \delta_{51}$	*vārīni*
	8: singular	x_g	*vāre*
	3: singular	$x + \delta_{28}$	*vārinā*
	3: dual, 4: dual, 5: dual	$x + \delta_{10}$	*vāribhyām*
	3: plural	$x + \delta_{11}$	*vāribhiḥ*
	4: singular	$x + \delta_{59}$	*vārine*
	4: plural, 5: plural	$x + \delta_{16}$	*vāribhyaḥ*
	5: singular, 6: singular	$x + \delta_{33}$	*vārinaḥ*
	6: dual, 7: dual	$x + \delta_{48}$	*vārinoḥ*
	6: plural	$x' + \delta_{23}$	*vārīnām*
	7: singular	$x + \delta_{51}$	*vārini*
	7: plural	$x + \delta_{25}$	*vāriṣu*
śuci	1: singular, 2: singular, 8: singular	x	*śuci*
	1: dual, 2: dual, 8: dual	$x + \delta_{52}$	*śucinī*
	1: plural, 2: plural, 8: plural	$x' + \delta_{51}$	*śucīni*
	8: singular	x_g	*śuce*
	3: singular	$x + \delta_{28}$	*śucinā*
	3: dual, 4: dual, 5: dual	$x + \delta_{10}$	*śucibhyām*
	3: plural	$x + \delta_{11}$	*śucibhiḥ*
	4: singular	$x + \delta_{59}$	*śucine*
	4: singular	$x_l + \delta_{61}$	*śucaye*
	4: plural, 5: plural	$x + \delta_{16}$	*śucibhyaḥ*

X	Vibhakti: number	Formula	Inflected forms
śuci	5: singular, 6: singular	$x + \delta_{33}$	*śucinaḥ*
	5: singular, 6: singular	$x_g + \delta_1$	*śuceḥ*
	6: dual, 7: dual	$x + \delta_{48}$	*śucinoḥ*
	6: dual, 7: dual	$x_y + \delta_{19}$	*śucyoḥ*
	6: plural	$x' + \delta_{23}$	*śucīnām*
	7: singular	$x + \delta_{51}$	*śucini*
	7: singular	$x_l + \delta_2$	*śucau*
	7: plural	$x + \delta_{25}$	*śuciṣu*
dadhi	1: singular, 2: singular, 8: singular	x	*dadhi*
	1: dual, 2: dual, 8:dual	$x + \delta_{52}$	*dadhinī*
	1: plural, 2: plural, 8: plural	$x_d + \delta_{51}$	*dadhīni*
	8: singular	x_g	*dadhe*
	3: singular	$x' + \delta_9$	*dadhnā*
	3: dual, 4: dual, 5: dual	$x + \delta_{10}$	*dadhibhyām*
	3: plural	$x + \delta_{11}$	*dadhibhiḥ*
	4: singular	$x' + \delta_{13}$	*dadhne*
	4: plural, 5: plural	$x + \delta_{16}$	*dadhibhyaḥ*
	5: singular, 6: singular	$x' + \delta_3$	*dadhnaḥ*
	6: dual, 7: dual	$x' + \delta_{19}$	*dadhnoḥ*
	6: plural	$x' + \delta_{20}$	*dadhnām*
	7: singular	$x' + \delta_{22}$	*dadhni*
	7: singular	$x_l + \delta_{60}$	*dadhani*
	7: plural	$x + \delta_{25}$	*dadhiṣu*
suvāc	1: singular, 2: singular, 8: singular	x'	*suvāk*
	1: dual, 2: dual, 8: dual	$x + \delta_{58}$	*suvācī*
	1: plural, 2: plural, 8: plural	$x_l + \delta_{57}$	*suvāñci*
	3: singular	$x + \delta_9$	*suvācā*
	3: dual, 4: dual, 5: dual	$x'_s + \delta_{10}$	*suvāgbhyām*
	3: plural	$x'_s + \delta_{11}$	*suvāgbhiḥ*
	4: singular	$x + \delta_{13}$	*suvāce*
	4: plural, 5: plural	$x'_s + \delta_{16}$	*suvāgbhyaḥ*
	5: singular, 6: singular	$x + \delta_3$	*suvācaḥ*
	6: dual, 7: dual	$x + \delta_{19}$	*suvācoḥ*
	6: plural	$x + \delta_{20}$	*suvācām*
	7: singular	$x + \delta_{22}$	*suvāci*
	7: plural	$x' + \delta_{25}$	*suvākṣu*

X. OPTIMISATION OF THE COMPUTATIONAL MODEL

The above computational model developed and implemented using XML structures, can be optimized in view of the consideration that the requirement is only to find the inflected forms of words that are needed for a comprehensive word-search.

Firstly, since the '+' operation in the formulae represents string concatenation, formulae of the types $x + \delta_i$ and $x + \delta_i + \delta_j$ contain the original word X as a substring. It is therefore sufficient to include only those transformations in the list that bring about some change in the word other than a mere appending of a suffix. For instance, when a search for the word '*sītā*' is sought to be done, the forms '***sītā***bhiḥ', '***sītā***bhyām', '***sītā***bhyaḥ', '***sītā***yāḥ', '***sītā***yām', etc. would automatically be identified. However, this search would not lead to the identification of the instances '*sīte*' or '*sītayā*' in the text. Hence, the formulae containing the latter at least as a substring, have to be retained. Similar is the argument when the word X' is being searched for in a text.

In the light of the above example, reducing the search words as detailed above may suggest another possibility at this stage, and that is to search for the word X after performing the operation x_l. In the above example, t

his would reduce the search list to '*sīt*', in which case this search word alone is sufficient, because all inflected forms of '*sītā*' contain '*sīt*' as a substring. However, it must be recognized that this is true in the case of this example but is certainly not the rule. The example of the word '*śvan*' (meaning 'dog') cited earlier is just one example of numerous such word categories where the inflected forms are so completely different from the given word X or even the most optimally chosen X'.

Further, the formula $x_l + \delta_i$ is different for different values of i, and hence there cannot be a reduction of this to just x_l.

In summary, if the formulae $x + \delta_i$ and $x + \delta_j$ are part of the list, then x alone is retained in the optimized model. Similar is the case with other forms derived from x, such as x', x_r and x_n, the only exception being x_l.

ALGORITHM 4. XmlStructureFeminineOptimized

```
<Gender G = "feminine">
    <LastLetter L = "ā">
        x, x′, x_r
    </LastLetter>
    <LastLetter L = "i">
        x, x′, x_d, x_g, x_ga, x_l + δ_2
    </LastLetter>
</Gender>
```

In the light of the above analysis, the XML structure presented in Algorithm 2 for feminine sample words is reduced to what is shown in Algorithm 4. Clearly, the same formulae hold for both categories of feminine words ending in '$\bar{a}$' in this optimized version. Also, the number of formulae has drastically reduced.

The number of computations required for both methods vis-à-vis feminine words, are depicted in Fig. 1. It was found that the average reduction in the number of computations in the optimized model is 83.5%. Thus, on average, more than 83% improvement in efficiency has been attained through the optimisation for word search in the case of feminine nouns.

The optimized XML structure for the neuter gender noun samples presented in Algorithm 3 is given in Algorithm 5. Fig. 2 provides the corresponding performance improvement statistics.

ALGORITHM 5. XmlStructureNeuterOptimized

```
<Gender G = "neuter">
    <LastLetter L = "i">
        x, x′, x_g
        <Category C = "2">
            x_y, x_l + δ_2, x_l + δ_61
        </Category>
        <Category C = "3">
            x_l + δ_60, x_d
        </Category>
    </LastLetter>
    <LastLetter L = "c">
        x, x′, x_l + δ_57, x′_s
    </LastLetter>
</Gender>
```

It was found that there is an average increased efficiency of 77.2% due to the optimisation in the case of neuter gender words.

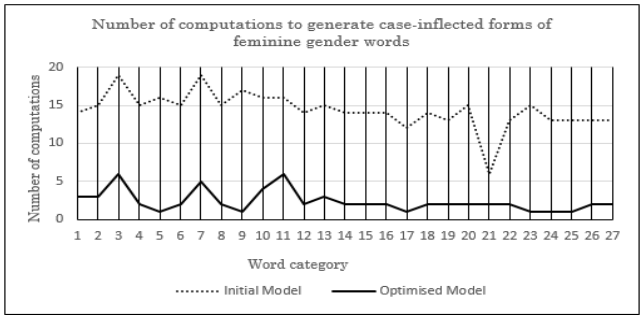

Fig. 1. Optimisation statistics for feminine gender *sādhāraṇa-śabdas*

For the sake of completeness we present here as Fig. 3, the optimisation statistics arrived upon earlier [19] on developing the optimized model for the 35 categories of masculine gender *sādhāraṇa-śabdas*. The average efficiency improvement is more than 80.6%. This optimisation technique having been devised, the search algorithm takes on two different dimensions as presented in the following section.

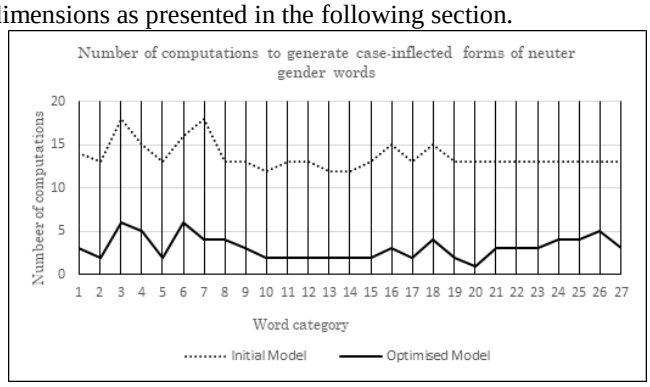

Fig. 2. Optimisation statistics for neuter gender *sādhāraṇa-śabdas*

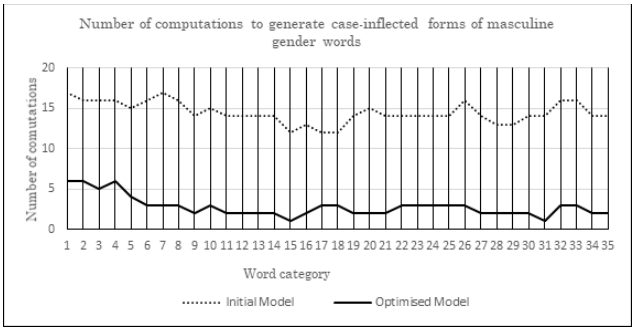

Fig. 3: Optimisation statistics for masculine gender *sādhāraṇa-śabdas*

XI. ENABLING DEEP AND SHALLOW SEARCHES

The above optimisation was done based on the assumption that word searches can result in the identification of the word not just as a whole word but also anywhere within larger words. This is what is, in general, accomplished by any search tool involving any language. In such searches, some instances identified may not exactly be what the user wants. For example, in English word search using any word processor such as Microsoft Word, searching for the word 'ant' in a text would yield not just the occurrences of the whole word, but also '**ant**elope', 'ten**ant**', '**cant**ankerous', etc. Many of these instances may not be what the user wants, and the user just skips such instances and proceeds to look at others.

With such a search, however, one can be sure that one has not missed any instance of the given word. And importantly, since there is no consideration of context, etc., the search results are obtained very quickly.

The search algorithms devised for Sanskrit are designed to perform substring matching by starting only from the beginning of words in the target text, and do not match substrings in the middle of words.

The optimized model presented above achieves a similar type of search result. So when the user seeks to search for the feminine word *rāmā* (meaning 'beautiful lady'), the optimized algorithm generates only the three truncated forms *rāmā, rāme* and *rāma*.

(The word '*rāmā*' has a declension table similar to the word '*sītā*'.) Searching for these instances would yield words such as '*rāmāṇām*' (masculine word meaning 'of the many *Rāmas*'), '*rāmāt*' (masculine meaning 'from Rāma') and '*rāmebhyaḥ*' (meaning 'for/to/from/than the many *Rāmas*') too, if these instances are present in the target text. Nevertheless, all that the user needs to do is to skip the instances he / she is not interested in, as is the way with general word searches. This is a fast search, though shallow, because the number of computations is almost 80% less.

However, the flip side to this type of search is that the instances identified may be incorrect and meaningless too. As seen in the above example, a search for a feminine gender word churns out masculine gender word forms as well. Such mix-ups occur very often across word genders. Thus, there may be irrelevant and even incorrect output that is generated.

If the user prefers a more meaningful search experience, then the algorithm that generates all unique inflected forms of the given word taking into account the word's gender, is the appropriate choice. The XML structures presented before optimisation, i.e. the ones in Algorithm 2 and Algorithm 3 are meant to generate inflected forms for this type of search, because they ensure that the complete word forms specific to the given gender, last letter and category of the word alone are generated. Since this involves more word forms to be searched for, there would be a small time lag for the program to actually produce the search results. However, since more meaningful search results are obtained, a slight reduction in speed would be acceptable to such a user. At any rate, as seen, the method of computation has been simplified to such an extent that this search does not cause very visible time lags.

XII. THE OVERALL SOLUTION TO THE WORD SEARCH PROBLEM IN SANSKRIT

Two models to generate case-inflected forms of a word have been presented above, in the context of their use in shallow and deep searching. However, there is another aspect that has to be handled in order for the deep search to yield truly meaningful results.

As discussed, giving the search input as '*sītā*, feminine', yields case-inflected forms such as '*sītābhiḥ*' and '*sīte*'. Now the word '*sītābhiḥ*' (meaning 'with the many Sītas') might occur in the text as '*sītābhir*' as a result of the operation of a *sandhi* rule. The non-optimized algorithm for deep search presented above would not recognize this instance. Similarly, the *s*-ending feminine word '*bhās*' (meaning 'light') would have forms such as '*bhābhyaḥ*' generated by the non-optimized algorithm, and a search for this word would miss out on occurrences of the word in the forms '*bhābhyas*', '*bhābhyo*', '*bhābhyaś*', etc. caused again by *sandhi* transformations. Similarly, internal sandhi transformations also need to be carried out for some of the words like '*vāri**n**ī*' shown in Table-IX to get corrected to the proper form, which is '*vāri**ṇ**ī*' in this case.

To handle these issues, we pass all the case-inflected forms of the given word into the *Sandhi* Engine developed earlier [20]. 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 because more search words are included, but that is an inevitable and small price to pay to gain a more effective and thorough search output.

The overall solution schematic for the computational model that solves the comprehensive word-search problem in Sanskrit is presented in Fig. 4.

This work hence assumes great importance in the light of the recent trend of trying to slow down searches to produce more meaningful results [28][29].

XIII. CONCLUSION

The problem of meaningful text-search in Sanskrit E-text has not been solved comprehensively in the literature till now. Such a solution could benefit scores of varieties of users apart form Sanskrit scholars, who wish to search for data in ancient Sanskrit texts that have been converted to E-text. A computational model that solves this problem has been comprehensively proposed by the authors in this and by a subset of the authors in earlier work [19][20].

Such a solution would require dealing with two syntactic phenomena in Sanskrit, viz. euphonic conjunctions and case-inflected forms. This paper presents a critical part of this comprehensive solution that involves the development of a computational model to generate case-inflected forms of feminine and neuter gender words in Sanskrit. All the categories of these nouns of the ordinary class in Sanskrit (*sādhāraṇa-śabdas*) have been studied and formulae for computing the case-inflected forms, developed. XML structures have been used to store the formulae hierarchically, and thus themselves double up as the algorithms to generate the word forms. Samples of such XML structures devised have also been presented. The generation of the case-inflected forms of these words has been done using a newly developed efficient algorithm by employing a novel pre-processing step, thus making the XML structures compact and quicker to parse.

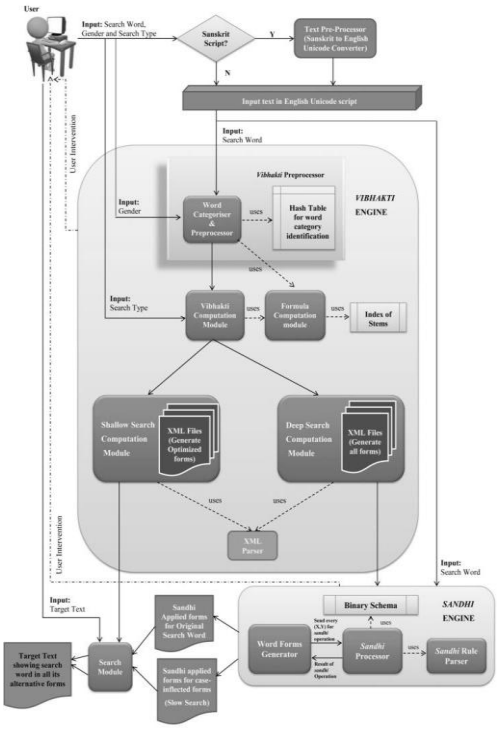

Fig. 4. The solution schematic for the word Search problem in Sanskrit.

Furthermore, two alternative search mechanisms for performing word searches on Sanskrit E-text have been presented in this work in the light of the models developed to generate the case-inflected forms of nouns.

One is a series of simple computations to generate all the unique case-inflected forms of a given word. These word forms are further processed and all those forms are derived, which could result from the possible euphonic conjunctions that could apply to them. [19] This entire set of words then forms the set of search words to be searched for in the target text. Though the model does make the generation of case-inflected forms simpler than existing models that take recourse to Pāṇini's rules directly or to more complex data structures and processing paradigms, the elaborateness of the exercise and the large number of search word forms generated could slow down the search. However, the flip-side is that more meaningful search results are obtained in this case.

On the other hand, an alternative optimized model developed here has been elaborated and a performance analysis presented. This optimized algorithm has been shown to be quicker than its parent algorithm, by 77-80%. However, this optimized algorithm serves to perform only a blind search of the text and could hence identify words in the target text that are not relevant in the context of the given word and its gender.

Thus, there is an inevitable trade-off between accuracy and speed in search algorithms, and this work has presented both the fast and the slow versions of algorithms for Sanskrit E-text search, to match different user preferences.

REFERENCES

1. Rick Briggs, "Knowledge Representation in Sanskrit and Artificial Intelligence", *RIACS, AI Magazine*, NASA Ames Research Center, 1985.
2. G. Cardona, "Pāṇini: his work and its traditions", *Motilal Barnasidass Publishers*, Delhi, India, 1988.
3. Anand Mishra, "Simulating the Pāninian System of Sanskrit Grammar", *Sanskrit Computational Linguistics: First and Second International Symposia*, ISBN: 978-3-642-00154-3, Springer-Verlag, 2009.
4. Donald E. Knuth, "Backus Normal Form vs. Backus Naur Form", *Communications of the ACM*, 7 (12): 735–736, doi:10.1145/355588 .365140, 1964.
5. Morrison and Kelly, "Backus Normal Form vs. Backus-Naur Form", *comp.compilers (newsgroup)*, 1993.
6. T. R. N. Rao and Subhash Kak, "The Panini-Backus Form in Syntax of Formal Languages", *Computing Science in Ancient India*, Munshiram Manoharlal, 2000.
7. P. Z. Ingerman "Panini-Backus form suggested", *Communications of the ACM*, 10:3, 137, doi:10.1145/363162.363165, 1967.
8. Akshar Bharati, Vineet Chaitanya and Rajeev Sangal, "Natural Language Processing, A Paninian Perspective", *Prentice Hall of India*, New Delhi, 1999.
9. Peter Scharf, "Levels in Pāṇini's Aṣtādhyāyi", *Proceedings, Third International Symposium on Sanskrit Computational Linguistics*, volume LNAI 5406, Springer, pages 66–77, 2009.
10. Göttingen Register of Electronic Texts in Indian Languages (GRETIL), http://gretil.sub.uni-goettingen.de/, 2001.
11. Mukthabodha Digital Library, http://muktalib5.org/ digital_library.htm, 2007.
12. Jon L. Bentley and Robert Sedgewick, "Fast algorithms for sorting and searching strings", *Proceedings of the 8th annual ACM-SIAM symposium on discrete algorithms*, ACM, 1997.
13. Ricardo Baeza-Yates and Gaston H. Gonnet, "A New Approach to Text Searching", *Communications of the ACM*, Volume 35, Issue 10 October, pages 74-82, 1992.
14. Amba Kulkarni et. al., "Building Morphological Analysers and Generators for Indian Languages using FST", *Tutorial at ICON 2010*, IIT Kharagpur, 2010.
15. Gérard Huet, "Design of a Lexical Database for Sanskrit", *COLING Workshop on Electronic Dictionaries*, Geneva, pages. 8-14, 2004.
16. Girish Nath Jha, et. al., "Inflected Morphology Analyzer for Sanskrit", *Sanskrit Computational Linguistics* 1 and 2, SpringerVerlag LNAI 5402, pages 219-238, 2009.
17. Smita Selot et. al., "Subanta pada analyzer for Sanskrit", *Oriental Journal of Computer Science & Technology*, Vol. 3(1), pages 89-93, 2010.
18. Shivamurthy Swamiji, "Ganakashtadhyayi", www. taralabalu. org /panini/ shabdarupa/, 2003.
19. Kasmir Raja S. V., Rajitha V., and Meenakshi Lakshmanan, "Computational Model to Generate Case-inflected Forms of Masculine Nouns for Word Search in Sanskrit E-text", *Journal of Computer Science*, 10 (11), ISSN 1549-3636, pages 2260-2268, 2014.
20. Rajitha V., Kasmir Raja S. V., and Meenakshi Lakshmanan, "Computational Algorithms Based on the Paninian System to Process Euphonic Conjunctions for Word Searches", *International Journal of Computer Science and Information Security*, ISSN 1947-5500, Vol. 12, No. 8, pages 64-76, 2014.
21. Naresh Jha, *"*Pāṇini's Aṣtādhyāyi*"*, *Chaukhamba Surbharati Prakashan*, Varanasi, 2004.
22. A. Goyal and Behra L. Kulkarni, "Computer Simulation of Aṣtādhyāyi: Some Insights", *Sanskrit Computational Linguistics* 1 and 2, Springer-Verlag LNAI 5402, pages 139-161, 2009.
23. B. S. Gillon, "Pāṇini's Aṣtādhyāyi and linguistic theory", *Journal of Indian Philosophy*, 35: 445– 468, 2007.
24. Bhaṭṭoji Dīkṣita, "Siddhānta-kaumudī", Translated by Śrīśa Candra Vasu, Volume 1, Motilal Banarsidas Publishers, Delhi, India, 1962.
25. A. Sharma, K. Deshpande, and D. Padhye, "Kāśikā: A Commentary on Pāṇini's Grammar". *Sanskrit Academy series*, Sanskrit Academy, Osmania University, 2008.
26. Vāmana and Jayāditya, "Kāśikā with the subcommentaries of Jinendrabuddhi, Haradatta Miśra and Dr. Jaya Shankar Lal Tripathi", Tara Book Agency, Varanasi, India, 1984.
27. Vidyasagar K. L. V. Sastry and Pandit L. Anantarama Sastri, "Śabdamañjarī", R. S. Vadhyar & Sons Publishers, Palghat, India, revised edition, 2002.
28. Jaime Teevan, Kevyn Collins-Thompson, Ryen W. White and Susan Dumais. "Viewpoint: Slow Search", DOI:10.1145/2633041, *Communications of the ACM*, Vol. 57, No. 8, 2014.

29. Christina Aperjis, Bernardo A. Huberman, and Fang Wu, "Human speed-accuracy tradeoffs in search", http://www.hpl.hp.com/research/scl/ papers / speedaccuracy / speedaccuracy.pdf, 2010.

AUTHORS PROFILE

Rajitha V has received Doctorate Degree in Computer Science in 2017 from Mother Teresa Women's University, Kodaikanal, India. Her area of interest include Natural Language Processing, digital electronics and operating systems. She is currently working as Assistant Professor in the Department of Computer Science, Meenakshi College for Women, Chennai, India.

Suganya R has received Doctorate Degree in Computer Science in 2011 from Mother Teresa Women's University, Kodaikanal, India. She has 10 years of research experience, and her research interests include cryptography and power optimization. She is currently working as Assistant Professor in the Department of Computer Science, Meenakshi College for Women, Chennai, India.

Meenakshi Lakshmanan heads the Department of Computer Science, Meenakshi College for Women, Chennai, India. She received her Doctorate Degree in Computer Science in 2011 from Mother Teresa Women's University, Kodaikanal, India. She has more than 10 years of research experience and has over 10 publications in International peer-reviewed journals and conferences. She is a member of the ACM. Her major research area is Natural Language processing with special reference to Sanskrit language.