Comparing Sanskrit Texts for Critical Editions: the sequences move problem.

Nicolas Béchet[1] and Marc Csernel[2,3]

[1] GREYC Université de Caen Basse-Normandie
Campus II science 3
14032 Caen CEDEX, France.
nicolas.bechet@unicaen.fr
[2] INRIA, Projet AXIS,
Domaine de Voluceau, Rocquencourt BP 105
78153 Le Chesnay Cedex, France
Marc.Csernel@inria.fr
[3] Universitée Paris Dauphine
1 Place du Maréchal Delattre de Tassigny
75016 Paris, France.

Abstract. A critical edition takes into account various versions of the same text in order to show the differences between two distinct versions, in terms of words that have been missing, changed, omitted or displaced.
Traditionally, Sanskrit is written without spaces between words, and the word order can be changed without altering the meaning of a sentence.
This paper describes the characteristics which make Sanskrit text comparisons a specific matter. It presents two different methods for comparing Sanskrit texts, which can be used to develop a computer assisted critical edition.
The first one method uses the L.C.S., while the second one uses the global alignment algorithm. Comparing them, we see that the second method provides better results, but that neither of these methods can detect when a word or a sentence fragment has been moved.
We then present a method based on N-gram that can detect such a movement when it is not too far from its original location.
We will see how the method behaves on several examples and look for future possible developments.

1 Introduction

A critical edition is one that takes into account all the different known versions of the same text. If the text is mainly known through a great number of manuscripts that include non trivial differences, the critical edition often looks rather daunting for readers unfamiliar with the subject: the edition is mainly made up by footnotes that highlight the differences between manuscripts, while the main text (that of the edition) is rather short, sometimes just a few lines per page. The differences between the texts are usually described in terms of words (sometimes sentences) that are missing, or have been added or changed in a specific manuscript. This bring up to mind the edit distance but in term of words rather than characters. The text of the edition is established by the

editor according to his own knowledge of the text. It can be a particular manuscript or a 'mean' text built according to some specific criteria. Building a critical edition by comparing texts one with another, especially manuscript ones, is a task which is certainly long and, sometimes, tedious. This is why, computer programs have long been helping philologists in their work (see [10] or [9] for example), but most of them are dedicated to texts written in Latin (sometimes Greek) scripts.

In this paper we focus on the problems involved by a critical edition of manuscripts written in Sanskrit. Our approach is illustrated by texts that are extracted from manuscripts of the "Banaras gloss": the *kāśikāvṛtti*.

The Banaras gloss was written around the 7th century A.D., and is one of the most famous commentaries on Pāṇini's grammar, which is known as the first **generative** grammar ever written, and was written around the fifth century B.C. as a set of rules. These rules cannot be understood without the explanations provided by a commentary such as the *kāśikāvṛtti*. This corpus was chosen, because it is one of the largest collection of manuscripts (about hundred different ones) of the same text actually known. Notice that, since some manuscripts have been damaged by mildew, insects, rodents, etc.., they are not all complete.

In what follows we first describe the characteristics of Sanskrit that matter for text comparison algorithms, we will then show that such a comparison requires the use of a lemmatized text as the main text. Using a lemmatized text induces the need of a lexical preprocessing. Once the lexical preprocessing has been carried out, we can proceed to the comparison, where two approaches have been developed, the first one based on the Longest Common Subsequence (L.C.S.) by [4], and the second one on edit distance by [3]. The second method is easier to use, so we use it to align our Sanskrit texts before moving sequences.

Because the word order is not always meaningful in Sanskrit, some manuscripts have some words sequences which are not in the same place than in the main text, and the alignment procedure, whichever it is, is not able to align such sequences together. The misplaced sequence appears to be missing in on place, and added in another one. This is why we use here a word n-gramm based method, to discover if some sequence moves are likely to exist, and then determine their precise limits, where they have been moved, and display them. This improvement is detailed in the section 4 of this paper. Remark that the sequence move problem is quite similar to the translocation problem which exists in genomics.

2 How to compare Sanskrit manuscripts

2.1 Sanskrit and its graphical characteristics

One of the main characteristics of Sanskrit is that it is not linked to a specific script. Here however we provide all our examples using the Devanāgarī script, which is nowadays the most most widely used. The script has a 48-letter alphabet.

Due to the long English presence in India, a tradition of writing Sanskrit with the Latin alphabet (a transliteration) has long been established and used by many European scholars such as Franz Bopp in 1816. All these transliteration schemes were originally

carried out to be used with traditional printing. It was adapted for computers by Frans Velthuis [12], more specifically to be used with TEX. According to the Velthuis transliteration scheme, each Sanskrit letter is written using one, two or three Latin characters; notice that all our corpus is written according to the Velthuis scheme and not in Devanāgarī Unicode [2].

In ancient manuscripts, Sanskrit is written without spaces, and this is an important graphical specificity, because it greatly increases the complexity of text comparison algorithms.

On the other hand, each critical edition deals with the notion of word. Since electronic Sanskrit lexicons such as the one built by Huet [6, 5] do not cope with grammatical texts, one must find a way to identify each Sanskrit word within a character string, without the help of either a lexicon or of spaces to separate the words.

The reader interested in exploring deeper approach of the Sanskrit characteristics which matter for a computer comparison can refer to [4].

2.2 How to proceed?

The solution comes from the lemmatization of one of the two texts of the comparison: the text of the edition. The lemmatized text is prepared **by hand** by the editor. It is called *padapāṭha*, according to a mode of recitation where syllables are separated. ¿From this lemmatized (the *padapāṭha*) text, we will build the text of the edition, which is called *saṃhitapāṭha*, according to a mode of recitation where the text is said continuously. The transformation of the *padapāṭha* into the *saṃhitapāṭha* is not straightforward because of the existence of *sandhi* rules.

What is called *sandhi* — from the Sanskrit: liaison — is a set of phonetic rules which apply to the morpheme junctions inside a word or to the junction of words in a sentence. These rules are perfectly codified in Pāṇini's grammar. Roughly speaking, written Sanskrit reflects (via the *sandhi*) the liaison(s) which are made by a human speaker. A text with separators (such as spaces) between words, can look rather different (the letter string can change greatly) from a text where no separator is found.
An example of *padapāṭha*:

```
vi^ud^panna_ruupa_siddhi+v.rtti+iya.m kaa"sikaa_naama
```

We can see that words are separated by spaces and three different lemmatization signs: +, _, ^.

The previous *padapāṭha* becomes the following *saṃhitapāṭha*:

```
vyutpannaruupasiddhirv.rttiriya.mkaa"sikaanaama
```

the bold letters represent the letters (and the lemmatization signs) which have been transformed, according to a *sandhi* rule.

We call the typed text, corresponding to each manuscript: *Typed Manuscript (T.M.)*. Each *T.M.* contains the text of a manuscript and some annotation commands. The annotation commands keep trace of all the modifications of the manuscript not explicitly present in the text, such as change of ink color, a hole made by a rodent, etc.. They provide a kind of meta-information. Each manuscript is typed by a scholar.

The processing is done in four steps, but only two of them will be considered in this paper:

- **First step**: A lexical preprocessing. The *padapāṭha* is transformed into a virtual *saṃhitapāṭha* in order to make a comparison with a *T.M.* feasible.
  The transformation consists in removing all the separations between words and then in applying the *sandhi*. This virtual *saṃhitapāṭha* form the text of the edition.
- **Second step:** An alignment of a *T.M.* and the virtual *saṃhitapāṭha*. The aim is to identify, as precisely as possible, the words in the *T.M.*, using the *padapāṭha* as a pattern.
- **Third step:** Once the alignment has been achieved and the words of the *T.M.* have been determined, try to improve the alignments results. Determine which word have been added suppressed changed , or moved.
- **Fourth step:** : Display the results in a comprehensive way for the editor. This step is accomplished using XML.

The comparison is done paragraph by paragraph.

2.3 Why not use the `diff` algorithm

The very first idea to compare Sanwkrit text was to use `diff` in order to obtain the differences between two Sanskrit sequences.

But the results related in [4] were quite disappointing. The classical `diff` command line provided no useful information at all. The result of the comparison of the two following sequences: `"srii ga.ne"saaya nama.h` and `tasmai "srii_gurave namas` just said that they were different.

This is why [4] started to implement their own L.C.S. based algorithm. Its results appear in the right-hand column of Table 1. We can see that they are expressed in term of words.

```
1c1
< "sriigane"saayanama.h
---
> tasmai"sriiguravenama.h
```

`diff` without space

```
1d0
< tasmai
4c3,5
< gurave
---
> gane
> "
> saaya
```

`ediff` with space

```
Word 1 'tasmai' is :
- Missing
Word 2 '"srii' is :
- Followed by Added word(s)
'ga.ne"saaya'
Word 3 'gurave' is :
- Missing
```

L.C.S. based results without space

Table 1. Example of results with L.C.S.

2.4 The L.C.S based method

This method was developed by [4], and was the first method used to build critical edition of Sanskrit texts. The L.C.S matrix associated with the previous result can be seen in figure 1(p. 5). In this figure the vertical text represents the *saṃhitapāṭha*, the horizontal

text is associated with the *T.M.*. The horizontal bold dark lines were provided by the *padapāṭha*, before it was transformed into the *saṃhitapāṭha*.

The rectangles indicate how the correspondences between the *saṃhitapāṭha* and the *T.M.* were done. One corresponds to a missing word (`tasmai`) two correspond to a word present in both strings the words (`s"rii` and `nama.h`), and the last one corresponds to a word with a more ambiguous status, we can say either that the word has been replaced or that one word is missing and another word has been added.

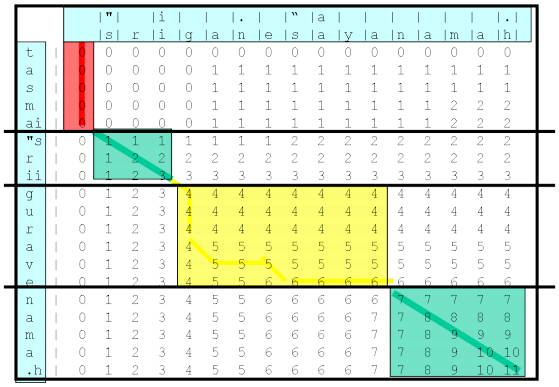

Fig. 1. The L.C.S. Matrix

If the result appears quite obvious within this example, it is not always so easy, particularly when different paths within the matrix can lead to different alignments providing different results. This led the authors to look for some navigation rules which are precisely related in [4].

3 The alignment based on edit distances

This method has been developed by [3] because the L.C.S based method needed too much post processing to be really efficient and because the lack of ”substitution” relative to L.C.S method was really a handicap when just one or two letters were changed in a word. The method is still not perfect but is a better base than L.C.S. to provide in the simplest way good alignments, but some improvements still need to be done before this method can be used effectively. Building an alignment using the edit distance matrix, starts by the low right-hand corner, going to the upper left. So the letters of the of the *T.M.* are aligned on the first letter of the *saṃhitapāṭha* with the same value they meet coming from the end of the paragraph. Sometimes this procedure induces the presence of Orphan letters which will be moved to obtain a better alignment as explained in the next subsection .

3.1 Shifting the orphan letters

We call an orphan letter an isolated a letter belonging to an incomplete word of (usually) a manuscript. To obtain a proper alignment, these letters must fit with the words to which they belong.

v	a	r	n	aa	n	aa	m	u	p	a	d	e	"s	a	.h	p	r	a	t	y	aa	h	aa	r	aa	r	th	a	.h	p	r	a	t	y	aa	h	aa	r	o
v	a	r	n	aa	n	aa	m	u	p	a	d	e	"s	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**a**	**.h**	p	r	a	t	y	aa	h	aa	r	o

Table 2: An example of orphan letters

v	a	r	n	aa	n	aa	m	u	p	a	d	e	"s	a	.h	p	r	a	t	y	aa	h	aa	r	aa	r	th	a	.h	p	r	a	t	y	aa	h	aa	r	o
v	a	r	n	aa	n	aa	m	u	p	a	d	e	"s	**a**	**.h**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	p	r	a	t	y	aa	h	aa	r	o

Table 3: Orphan letters after being shifted

Table 2 gives a good example. The upper line of the table represents the *padapāṭha*, the lowest one a *T.M.*. The word separation induced by the *padapāṭha* are indicated by double vertical lines. Because the *padapāṭha* is used as a template the separations appears also within the *T.M.*. The orphan letters appears in bold. The words `pratyaahaaraa` and `rtha.h` are missing in the *T.M.*. Consequently the letters `a.h` are misplaced, with the word `rtha.h`. The goal is to shift them to the right place with the word `upade"sa.h`. The result after shifting the letters appears in Table 3. The bold letters are the letters which has been shifted.

In the second example (Table 4) we see on the left side of the table that the letter `a` must just be shifted from the beginning of `asyddhy` to the end of `saavarny` giving the right-hand part of the table.

s	aa	v	a	r	.n	y	a	p	r	a	s	y	d	dh	y
s	aa	v	a	r	.n	y	-	-	-	**a**	s	y	d	dh	y

The orphan letters

s	aa	v	a	r	.n	y	a	p	r	a	s	y	d	dh	y
s	aa	v	a	r	.n	y	**a**	-	-	-	s	y	d	dh	y

After being shifted

Table 4: Another example

Another kind of possible shift is the one linked to the presence of supplementary letters within the *T.M.* such as appears in the left part of table 5. The letters `a` and `nam` of the *padapāṭha* are shifted to the left of the word as appears in the right-hand part.

p	r	a	y	o	j	-	-	-	-	-	**a**	-	-	-	-	**n**	**a**	**m**
p	r	a	y	o	j	a	n	a	m	s	a	.m	j	"n	aa	n	a	m

Before shifting

p	r	a	y	o	j	**a**	**n**	**a**	**m**	-	-	-	-	-	-	-	-	-
p	r	a	y	o	j	a	n	a	m	s	a	.m	j	"n	aa	n	a	m

After being shifted

Table 5: Shitfing within the *padapāṭha*

3.2 Measuring the quality of the alignment.

It is difficult to find a unique method to measure the quality of the alignments, because each method (L.C.S. and Edit Distance) produces a different type of alignment where the improvement must be made in a different way. L.C.S. methods do not induce any substitution, so measuring of the quality is quite easy to find: more "empty" characters we have, the lower the quality is; even if it can be nuanced in some particular cases. On the other hand with the edit distance based method, the algorithm provides substitutions and we can observe that the more substitutions we have, the lower quality. Remark that all substitutions are not irrelevant, and should not lower the quality, but some of them do, mainly the one corresponding to a possible translocation. Missing or added letters obtained by the edit distance method have always been relevant. Because an important part of the substitution are relevant we will never obtain a null score. We know we have to improve our measure

4 An improvement of the alignment by an n-gram based method.

The goal of this approach is to improve the alignment (obtained by edit distance method) of the *saṃhitapāṭha* and a *T.M.* once the orphan letters have been shifted.
An example of an alignment which can be improved is written in the table 6. The bold letters are letters which have been substituted, and lowered the quality.

n	**u**	**b**	-	a	**n**	**dh**	-	a	.h	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**	a	n	u	n	aa	s	i	k	a	.h
r	**e**	**t**	v	a	**k**	**aa**	r	a	.h	**i**	**t**	**s**	**a**	**.m**	**j**	**~n**	**o**	a	n	u	n	aa	s	i	k	a	.h

Table 6: An alignment with a low score

We define the alignment ***score*** by the number of substituted letters, better the alignment, lower the score. We obtain with this alignment a score of **13**, which is poor.
This score can be improved if we refer to the complete sentence in the *saṃhitapāṭha* and the *T.M.*: we can then move the sequence `la kaare tv` of the *T.M.* by inserting empty letters on the left (*i.e.* the '-' symbol), in the *T.M.* and on the right in the *saṃhitapāṭha* we obtain a good alignment (*with a score of 0*[4]) by aligning `la kaare tv` in a manuscript with `la kaare tv` of the *saṃhitapāṭha* as in the alignment below in table 7. The bold letters indicate the sequence moved.

n	u	b	-	a	n	dh	-	a	.h	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	a	n	u	n	aa	s	i	k	a	.h
-	-	-	-	-	-	-	-	-	-	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**	a	k	aa	r	a	.h	i	t	s	a	.m	j	~n	o	a	n	u	n	aa	s	i	k	a	.h

Table 7: The alignment improved

[4] Note that an insertion (noted '-') is not lowering the score with the edit distance alignment.

4.1 Overview of the procedure

It is described by the steps below, which are detailed in the next subsections.

1. We first extract word n-gramfrom the *saṃhitapāṭha* (cf § 4.2).
2. We search for each word n-gram all the possible better alignments (called a *candidate*) in the *T.M.*
3. For each *candidate*, we modify the original alignment to take the *candidate* into account, and build new word borders according to the *saṃhitapāṭha* (cf § 4.3).
4. We apply optimizations to the new alignment (cf § 4.4).
5. For each new generated alignment, we build a new score and write a proposition of improvement.

4.2 Extracting the word n-gram

The first step is to extract ***word n-gram***. Word n-gram are frequently employed in the literature, one of the first uses being [11], or more recently in [8]. An *n*-gram of *X* can be defined as a sequence of *n* successive *X* with $\boldsymbol{X} \in \{\boldsymbol{word}, \boldsymbol{letter}\}$. The figure 2 provides an example of letter 2-gram and word 3-gram extraction .

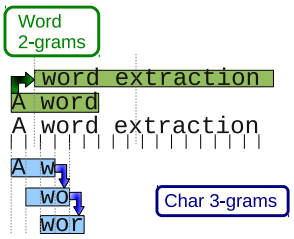

Fig. 2. Example of n-gram extraction.

We use word n-gram extraction in the hope to place all occurrences of a word in a unique context. A single word can have many occurrences in the *saṃhitapāṭha*, but only one when it is preceded and followed by some specific words. The word `kaare` in table 8 has a lot of instances but only one which is preceded by word `la` and followed by the word `tv`. Thus, by taking a 3-gram, we obtain only one instance of the word 3-gram `la kaare tv` and our extraction becomes relevant.

We use the extraction to make a (word) segmentation of the *saṃhitapāṭha*. An example of 4-gram from the *saṃhitapāṭha* is given in the upper line of Table 8, the lower line contains letters which do not fit the 4-gram, and that we will try to move, the bold letters are well aligned. In the following when we talk about *word n-gram*, we always refer to the *saṃhitapāṭha*.

n	u	b	-	**a**	n	dh	-	**a**	**.h**	l	a	k	aa	r	e	t	v
r	e	t	v	**a**	k	aa	r	**a**	**.h**	i	t	s	a	.m	j	˜n	o

Table 8: 4-gram alignment example with a score **13**.

The *n* value is set by the user, but could, in the future, be computed by program.

4.3 The search for candidates

Candidates are defined as strings in a manuscript which could provide a better alignment for a given word n-gram. Thus, *candidates* must contain as many letters than the word n-gram. In figure 3, the word n-gram `nub-andh-a.h la kaare tv` has **16**

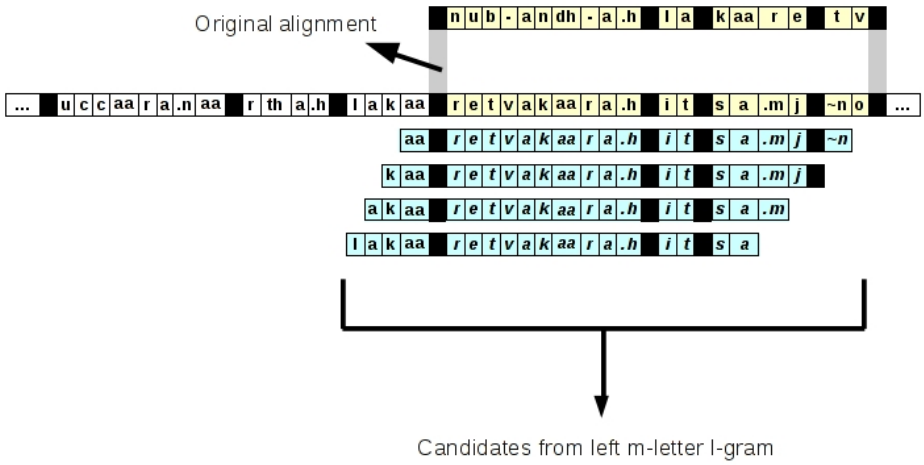

Fig. 3. The search for candidates for the word 4-gram `nub-anh-ah la kare tv`.

letters (without spaces). Then, all candidates must have **16** letters. To get them we shift a sliding window of **16** letters, from left to right, and right to left from the initial position.The number of shifting is given by the parameter $\boldsymbol{m}$; in the example shown in figure 3. ,the parameter $\boldsymbol{m = 4}$: we have shifted the sliding window of **4** positions to the left [5]

To estimate the new alignment quality we compute the score, as defined in section 4: the number of substitutions made between the two strings (the one from the *saṃhitapāṭha* is the current word n-gram, the other one from the *T.M.* is a *candidates*. The alignment of strings: `nub-andh-a.h la kaare tv` and `retvakaara.h it sa.mj ~no` in Table 8 obtain, for instance, a score of **13**.

If a *candidate* provides a better score that the original one, we keep it. At the end, we keep the best, the on with the lowest score. If several *candidates* have the same score, we keep the last one, but this situation never occurred with the examples we have considered.

[5] An *m* value of 4 means that 8 candidates are tested, 4 from left m-letter l-gram and 4 from right m-letter l-gram

To summarize, we first need to compute the score of the original alignment (**13**). Then, the original word n-gram `nub-andh-a.h la kaare tv` is compared with the 4 *candidates*:

- `aaretvakaara.hit sa.mj˜n`,
- `kaaretvakaara.hitsa.mj`,
- `akaaretvakaara.hitsa.m`,
- `lakaaretvakaara.hitsa`

The word separator ' ' is not reported because word separation is no longer relevant (*i.e.* words are no longer aligned). At the end, with these four *candidates*, none of the scores is better than **13**, the four *candidates* must be rejected.

4.4 Integrating the best candidates

After the selection of the best *candidate* for a given word n-gram, we need to build new word boundaries. Let us consider the example in table 8 with the word n-gram `nub-andh-a.h la kaare tv`. With the parameter m = 20, we find `caara.naartha.hlakaaretv` as the best *candidate*. To align this *candidate* with the word n-gram we need first to adjust the corresponding word boundaries.

Aligning the n-gram and the candidate. In the following tables, the n-gram and it corresponding best *candidate* are in bold:

u	c	c	aa	r	a	.n	aa	r	th	a	.h	n	-	-	aa	**n**	**u**	**b**	**-**	**a**	**n**	**dh**	**-**	**a**	**.h**	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**

Table 9: the word 4-gram in the *saṃhitapāṭha*

k	aa	r	a	u	c	**c**	**aa**	**r**	**a**	**.n**	**aa**	**r**	**th**	**a**	**.h**	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**	a	k	aa	r	a	.h

Table 10: the best candidate in a *manuscript*

We need to adjust the two texts to align `nub-andh-a.h la kaare tv` with `caara.naartha.hlakaaretv`. This adjustment is made by adding insertions of empty characters both in the *saṃhitapāṭha* and the *T.M.*

The number of insertions is equal to the absolute value of the substraction of the position of the first letter of the word n-gram and the position of the first letter of the candidate. In our example, the first letter of the word n-gram is **24** and the position of the first letter of the candidate is **10**. Thus, we need to insert **14** letters. We obtain the alignment described in Table 11, in this table the dots represent the places where the insertions have been done. The number of dot is not relevant. In the sequence corresponding to a manuscript, the word limits have disappeared, because they are no more relevant, we will reconstruct them on the next step.

u	c	c	aa	r	a	.n	aa	r	th	a	.h	n	-	-	aa	**n**	**u**	**b**	**-**	**a**	**n**	**dh**	**-**	**a**	**.h**	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**	.	.	.	.	.	.	.

u	c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	**c**	**aa**	**r**	**a**	**.n**	**aa**	**r**	**th**	**a**	**.h**	**l**	**a**	**k**	**aa**	**r**	**e**	**t**	**v**	a	k	aa

Table 11: The alignment before the word limits reconstruction

The insertion made is relative to the position of the word n-gram compared to the position of the candidate. If the candidate precedes the word n-gram (as in our example), we first need to shift the *T.M.* and then the *saṃhitapāṭha*, otherwise we shift the *saṃhitapāṭha* first.

The *saṃhitapāṭha* words alignment. After the *candidates* has been correctly aligned with the word n-gram, we need to align the existing words of the word n-gram with the unsegmented *candidate*, according to the *saṃhitapāṭha* words. For example, with the example from Table 8, the new word segment appears in the table 12. The alignment represented is not a good one, he still has a score = 6, which can be ameliorated by the following step, the bold letter are the letters which has been substituted.

n	**u**	**b**	-	**a**	**n**	**dh**	-	a	.h	l	a	k	aa	r	e	t	v
c	**aa**	**r**	a	**.n**	**aa**	**r**	th	a	.h	l	a	k	aa	r	e	t	v

Table 12: The newly reconstructed word boundaries

We can now consider the new word boundaries and report them within the *T.M.*

4.5 Optimizing the new alignments

Making a new alignment on texts can induce new orphan letters. Thus, we apply the approach presented in Section 3.1 to obtain a better alignment. For instance, with our previous example, we identify many orphan letters as 'c', 'aa', 'r', etc. on the *T.M.* as shown below:

u	c	c	aa	r	a	.n	aa	r	th	a	.h	n	aa	n	u	b	-	a	n	dh	-	a	.h	l	a	k	aa	r	e	t	v	-	-	-	-	.
u	c	-	-	-	-	-	-	-	-	-	-	-	-	**c**	**aa**	**r**	**a**	**.n**	**aa**	**r**	**th**	**a**	**.h**	l	a	k	aa	r	e	t	v	a	k	aa	r	.

Table 13: Identification of the new orphan letters.

u	c	c	aa	r	a	.n	aa	r	th	a	.h	n	aa	n	u	b	-	a	n	dh	-	a	.h	l	a	k	aa	r	e	t	v	-	-	-	-	...
u	c	**c**	**aa**	**r**	**a**	**.n**	**aa**	**r**	**th**	**a**	**.h**	-	-	-	-	-	-	-	-	-	-	-	-	l	a	k	aa	r	e	t	v	a	k	aa	r	...

Table 14: The new orphan letters after a shift.

4.6 Building the final score

The final step of our approach is the final score of the new alignment computation, according to the whole environment (the neighboring words) and it delivery to an expert. The new computed score is quite different from the previous one used to select the candidates. The definition is the same: the number of substitutions made between the *saṃhitapāṭha* and the *T.M.* The difference lies in the texts we use for the comparison. For candidate selection, we only focus on a word n-gram and a *candidate*, which has the same number of letters. For the final score we use the new alignment.

4.7 Experiments

Experimental protocol. We use as a corpus the first chapter of the *kāśikāvṛtti*: the *Pratyāhrās sutra*, and we have used only 8 manuscripts (*T.M.*) randomly chosen from

the fifty available. Each manuscript contains a maximum of 34 paragraphs. We first apply the alignment method, then after shifting the orphan letters, we obtain a baseline alignment. Our goal is to improve the baseline by lowering the number of letter substitutions. Recall that with edit distance, the more letter substitutions we have, poorer the alignment, higher the score.
Then we apply our approach. Considering $\boldsymbol{P}$ paragraphs, $\boldsymbol{K}$ manuscript, $\boldsymbol{w}$ maximum words by paragraphs, which means w - (n-1) word n-gram by paragraphs, and $\boldsymbol{m}$ tests of possible candidates for each word n-gram, the maximum numbers of operation $\boldsymbol{Nb}$ to apply our method is approximated by:
$\boldsymbol{Nb = P \times K \times m \times (w - (n - 1))}$. We have a complexity depending mostly on the $\boldsymbol{n}$ of n-gram, the number of paragraphs, the number of manuscripts, and the number of possible candidates to be tested.

For instance, with the parameters P=34, K=8, n=4, m= 40, we obtain:
$\boldsymbol{Nb = 34 \times 8 \times 40 \times (w - (4 - 3)) = 10,880 \times (w - 3)}$.

n	*m*	*Nb*	*Score*
baseline			1,110
1 (words)	40	10,880 * l	922
2	40	10,880 * l	908
3	40	10,880 * (l – 2)	898
4	40	10,880 * (l – 3)	940
3	20	10,880 * (l – 2)	1,080
3	60	10,880 * (l – 2)	784
3	80	10,880 * (l – 2)	784

Table 15. Experimental results.

Experimental results.
The results are displayed in Table 15. The baseline obtains a score of 1,110. It means that 1,110 substitutions were made during the construction of the alignment of *saṃhitā-pāṭha* and the 8 different manuscripts.
We first want to see the influence of the $\boldsymbol{n}$ parameter. According to our experiments, the best score is obtained with $\boldsymbol{n = 3}$ allowing 212 substitutions to be suppressed from the baseline. However, the $\boldsymbol{n}$ parameter does not seam to have an important influence on the scores.
In comparison, a modification of $\boldsymbol{m}$ parameter provides a better score with only 784 substitutions. It seams that 326 substitutions were deleted compared with the baseline. Note that we obtain the same score for $\boldsymbol{m = 60}$ and $\boldsymbol{m = 80}$; $\boldsymbol{m = 60}$ is better because we need fewer operations to obtain the same quality of alignment.

5 Future work and Conclusion

5.1 Future Improvement

Our first trials made us discover some unexpected situations that we do not take into account. Consider the alignment produced by our program in table 16, where the bold

letters are orphan letters unusually placed in the *saṃhitapāṭha*. We need to shift them to get a better result which can be seen in table 17, where the italic letters refer to letters concerned by our second improvement.

a	t	r	a	d	v	i	r	v	a	c	a	n	a	m	p	r	aa	p	n	o	t	i	a	c	o	r	a	h	aa	bh	y	aa	.m	...
a	t	r	a	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	a	c	o	r	a	h	aa	bh	y	aa	.m	...

...	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**d**	**v**	**e**	**i**	-	-	**t**	**i**
...	d	v	e	i	t	i	d	v	i	r	v	a	c	a	n	a	m	p	r	aa	p	n	o	t	i

Table 16: An unexpected alignment .

Because of the bad alignment induced by the orphan letters in the *saṃhitapāṭha* our n_gram based method was unable to discover that the sequences in italic letters of figure 17 was corresponding to a translocation, and then did not made the move which lead us to the right alignment such as displayed in the table 18, where the bold letters correspond to the letters newly aligned.

a	t	r	a	*d*	*v*	*i*	*r*	*v*	*a*	*c*	*a*	*n*	*a*	*m*	*p*	*r*	*aa*	*p*	*n*	*o*	*t*	*i*	a	c	o	r	a	h	aa	bh	y	aa	.m	...
a	t	r	a	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	a	c	o	r	a	h	aa	bh	y	aa	.m	...

...	**d**	**v**	**e**	**i**	**t**	**i**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
...	d	v	e	i	t	i	*d*	*v*	*i*	*r*	*v*	*a*	*c*	*a*	*n*	*a*	*.m*	*p*	*r*	*aa*	*p*	*n*	*o*	*t*	*i*

Table 17: The alignment after shifting the orphan letters of the *saṃhitapāṭha*

To solve this problem, it seems at first sight that we need only to take a better care of the orphan letters in the *saṃhitapāṭha*, but we need a real trial to check if this simple action will be sufficient .

We can remark that the use of n-gram is not the only possible way to solve the translocation (sequence move) problem. A problem quite similar exists in genomics, and can be solved by methods such as ”**Glocal** alignment” [1]. These methods do not follow the same goal than we do, but they may be source of inspiration for our future work.

a	t	r	a	**d**	**v**	**i**	**r**	**v**	**a**	**c**	**a**	**n**	**a**	**m**	**p**	**r**	**aa**	**p**	**n**	**o**	**t**	**i**	a	c	o	r	a	h	aa	bh	y	aa	.m	d	v	e	i	t	i
a	t	r	a	**d**	**v**	**i**	**r**	**v**	**a**	**c**	**a**	**n**	**a**	**m**	**p**	**r**	**aa**	**p**	**n**	**o**	**t**	**i**	a	c	o	r	a	h	aa	bh	y	aa	.m	d	v	e	i	t	i

Table 18: New words alignment obtained by a sequence move.

5.2 Improvement of the score computation

The score we use provides good results according to our purpose. But we would like to get a criterion which can provide a value near from zero when the two texts compared differ only by the usual differences that one can expect when comparing two different versions of the same text. Two reach this goal we will have to go step by step to drop out some elements that we don’t want to take into account for the score.

For example some Sanskrit letters are prone to be confused by a scribe, so they need a special treatment, together for our score computation and in our alignment method. A

second step could be a special treatment for the substitution of sparse letters which is obviously due to a different cause than a long sequence substitution.

On the other hand we have to build a second critter which can be used with a L.C.S. If we succeed on both of these goal we will be able to compare the two approaches (L.C.S. and edit distance) with a measure.

5.3 Using the critical edition as a distance

While the definition of a critical edition reminds us the one of the edit distance, we can build a distance between two manuscripts using the *saṃhitapāṭha* as an intermediary. This will provide us the informations for the construction of a phylogenetic tree or Stemma codicum [7] between the different manuscript. The construction of Stemma codicum will be a great help for all philologists interested in Sanskrit texts.

5.4 Conclusion

We have presented a tool for computer assisted construction of a critical edition which provides results which are quite satisfactory. The tool is still not perfect an we have described in the previous paragraphs some of the possible amelioration.

But if we display our results with XML based tool, we can provide to the philologists a great help in the visualization of a critical edition allowing to chose dynamically which text should be compared and investigated. Sanskrit will have at last a tool which can be compared with the one existing for European languages.

References

1. M. BRUDNO AND AL: *Glocal alignment: finding rearrangements during alignment*, in ISMB (Supplement of Bioinformatics), 2003, pp. 54–62.
2. U. CONSORTIUM: *Unicode standard version 6.0: Devanagari*. http://unicode.org/charts/PDF/U0900.pdf, 2010.
3. M. CSERNEL AND T. CAZENAVE: *Comparing sanskrit texts for critical editions*, in COLING, Beijing, 2010, pp. 206–213.
4. M. CSERNEL AND F. PATTE: *Critical edition of sanskrit texts*, in Sanskrit Computational Linguistics, vol. 5402 of Lecture Notes in Computer Science, 2009, pp. 358–379.
5. G. HUET: *Design of a lexical database for sanskrit.*, in COLING Workshop on Electronic Dictionaries, Geneva, 2004, pp. 8–14.
6. G. HUET: *Héritage du sanskrit: Dictionnaire français-sanskrit*. http://sanskrit.inria.fr/Dico.pd, 2006.
7. M. LE POULIQUEN: *Using lattices for reconstructing stemma*, in Fifth International Conference on Concept Lattices and Their Applications, CLA., 2007.
8. H. LEI AND N. MIRGHAFORI: *Word-conditioned phone n-grams for speaker recognition.*, in Proc. of ICASSP, Honolulu, 2007.
9. C. MONROY ET AL.: *Visualization of variants in textual collations to analyse the evolution of literary works in the cervantes project.*, in Proceedings of the 6th European Conference, ECDL 2002, M. Agosti and e. Constantino Thanos, eds., Rome, Italy, September 2002, Springer, pp. 638–53.

10. P. O'HARA, R.J. ROBINSON: *Computer-assisted methods of stemmatic analysis.*, in Occasional Papers of the Canterbury Tales Project, N. Blake and P. Robinson, eds., vol. 1, Office for Humanities Communication, Oxford University, 1993, pp. 53–74.
11. R. L. SOLSO: *Bigram and trigram frequencies and versatilities in the english language.* In Behavior Research Methods & Instrumentation., 11(5) 1979, pp. 475–484.
12. F. VELTHUIS: *Devanāgarī for TEX, Version 1.2, User Manual*, University of Groningen., 1991, http://www.ctan.org/tex-archive/language/devanagari/velthuis/.