A Generalized LCS Algorithm and Its Application to Corpus Alignment

Jin-Dong Kim
Database Center for Life Science (DBCLS), ROIS, Japan
jdkim@dbcls.rois.ac.jp

Abstract

The paper addresses the problem of text variation which often hinders interoperable use or reuse of corpora and annotations. A systematic solution is presented based on a variation of Longest Common Sequence algorithm. An empirical experiment with 20 full text articles shows it works well with a real world application.

1 Introduction

Corpus with annotation is regarded indispensable for the development of natural language processing (NLP) technology. As so, many corpora with annotation have been developed, and many groups are working with new annotation projects.

As various annotated corpora accumulate in the field, reusability and interoperability is becoming an important issue (Cohen et al., 2005; Johnson et al., 2007; Campos et al., 2012). Among others, Wang et al. (2010) reports that there are a number of corpora that claim to have annotations for protein or gene names, e.g, Genia (Kim et al., 2003), Aimed (Bunescu et al., 2004), and Yapex (Franzén et al., 2002), and that, however, the protein annotations in those corpora are substantially different to each other, which calls for an interoperable interpretation of the annotations for integrative reuse of them. Rebholz-Schuhmann et al. (2011) investigates aggregation of variable named entity annotations in large scale, which also show the importance of interoperable use of corpus annotation.

There also have been efforts for the interoperability of corpora and annotations from a perspective of encoding and representation, e.g., Linguistic Annotation Framework (LAF) (Ide and Romary, 2004) and Open Linguistics[1]. Without a doubt, those efforts contribute to improving the interoperability of corpora and annotation.

[1] http://linguistics.okfn.org/

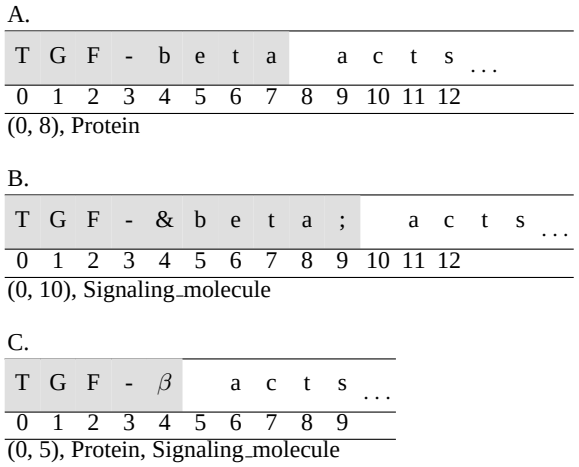

Figure 1: Text variations and annotation to them

This paper addresses another type of problem, *text variation*, which often hinders interoperable use or reuse of corpora and annotations in real world applications. As far as the author knows, it is the first attempt to develop a definite and systematic solution to the problem.

The problem of text variant is explained in detail in Section 2, while the solutions are presented in Section 3 and 4 After discussions on its real world application in Section 5, the paper is concluded in Section 6.

2 Task definition

Figure 1 illustrates a simple example of the problem to be addressed: A, B and C are text variants from the same document; The position index of the equivalent text spans, "*TGF-beta*", "*TGF-β*", and "*TGF-β*", are different to each other; And, the annotations made to the spans are not directly interoperable, although they are made to conceptually the same span of the same document.

Note that the example is extremely simplified for the ease if understanding. In reality, the problem is much more complex: a text, as the tar-

International Joint Conference on Natural Language Processing, pages 1112–1116,
Nagoya, Japan, 14-18 October 2013.

get of annotation, is often as long as hundreds, or thousands of characters, or even much longer, and a single local variant affects the entire remaining portion of the text, in a cumulative way.

Nowadays, the widespread use of Unicode is one of the reasons of text variant, particularly when it comes to text processing, because many NLP tools, e.g., syntactic parsers, cannot handle Unicode characters properly. Thus, during many annotation projects, Unicode characters, e.g., Greek letters, are spelled out into ASCII alphabets, like *beta* in Figure 1 A. Sometimes, extra symbols, e.g., '&' and ';', are inserted to delimit Unicode-origin sequences, like in B.

Suppose that two independent annotation projects took the text of C into their corpora, and their preprocessors spelled out Greek letters differently like in A and B. The projects may produce different annotations according to their interest and perspectives. While those annotations may serve their goals individually, further benefit, e.g., reuse, comparison, or aggregation, can be gained from interoperable use of them. In the example, we want the annotations, `(0, 8, Protein)`[2] from A and `(0,10, Signaling_molecule)` from B, to be transferable to C, or to each other. However, the variation of text poses a challenge: we need to compute the mapping between variations of text.

For standoff annotation, a text defines a one-dimensional Cartesian coordinate system, whereon any position on the text is specified. We thus cast the problem to the task of finding a mapping function from a one-dimensional Cartesian coordinate system to another, when they are filled with comparable values (characters). In Figure 2, $\delta_{A\to C}$ is a mapping function from A to B, which enables transferring the annotation to the source text. $\delta_{C\to A}$ is the mapping for the opposite direction. Once those functions are obtained, any annotation produced by the project A can be transferred to the original text, and vice versa.

3 LCS for text mapping

For most cases, text mapping can be computed using Longest Common Sequences (LCS) algorithms (Bergroth et al., 2000). LCS is a well known problem, based on which the UNIX command, *diff*, is implemented. Algorithm 1 is a dynamic algorithm to compute the length of LCS of any two strings. Figure 3 shows the resulted LCS table for example strings. From the LCS table, the *diff* between the two strings - see Figure 4[3] - can be read out by Algorithm 2.

[2]Throughout the paper, we make the span specification in the style of BioNLP shared task (Kim et al., 2009), where the beginning of a span is specified by the number of characters preceding the span, and the end by the number of characters up to the end of the span.

$\delta_{A\to C}$

0	1	2	3	4	5	6	7	8	9	10	11	⋯
0	1	2	3	4	4	4	4	5	6	7	8	

$\delta_{C\to A}$

0	1	2	3	4	5	6	7	8	9	⋯
0	1	2	3	4	8	9	10	11	12	

Figure 2: Mapping between text variations

		T	G	F	-	b	e	t	a		a	c	t	s
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	**1**	1	1	1	1	1	1	1	1	1	1	1	1
G	0	1	**2**	2	2	2	2	2	2	2	2	2	2	2
F	0	1	2	**3**	3	3	3	3	3	3	3	3	3	3
-	0	1	2	3	**4**	4	4	4	4	4	4	4	4	4
β	0	1	2	3	4	4	4	4	4	4	4	4	4	4
	0	1	2	3	4	4	4	4	4	**5**	5	5	5	5
a	0	1	2	3	4	4	4	4	4	5	**6**	6	6	6
c	0	1	2	3	4	4	4	4	4	5	6	**7**	7	7
t	0	1	2	3	4	4	4	4	4	5	6	7	**8**	8
s	0	1	2	3	4	4	4	4	4	5	6	7	8	**9**

Figure 3: The LCS table for "TGF-β acts" and "TGF-beta acts". The place of text variation is indicated in gray

Algorithm 1 LCS computation

```
1: function LCS(X[1..m], Y[1..n])
2:   C = ARRAY(0..m, 0..n)
3:   for i := 0..m do
4:     C[i, 0] := 0
5:   end for
6:   for j := 0..n do
7:     C[0, j] := 0
8:   end for
9:   for i := 1..m do
10:    for j := 1..n do
11:      if X[i] = Y[j] then
12:        C[i, j] := C[i − 1, j − 1] + 1
13:      else
14:        C[i, j] := MAX(C[i, j − 1], C[i − 1, j])
15:      end if
16:    end for
17:  end for
18: end function
```

Algorithm 1 has time and space complexities of $O(mn)$, where m and n are the length of the strings. For many real world applications, Hunt-

[3]In the first column, the minus ('-') and plus ('+') signs indicate *deletion* and *insertion* operations, respectively.

Algorithm 2 Reading out *diff* from LCS table

```
 1: D = STACK
 2: i := m
 3: j := n
 4: while i ≠ 0 or j ≠ 0 do
 5:   if i > 0 and j > 0 and X[i] = Y[j] then
 6:     PUSH(D, ['=', X[i], Y[j]])
 7:     i := i − 1
 8:     j := j − 1
 9:   else if j > 0 and ( i = 0 or C[i, j-1] > C[i-1, j]) then
10:     PUSH(D, ['+', nil, Y[j]])
11:     j := j − 1
12:   else
13:     PUSH(D, ['−', X[i], nil])
14:     i := i − 1
15:   end if
16: end while
```

Mcllroy algorithm (Hunt and McIlroy, 1976) is frequently used, which regularly beats the complexities of the dynamic algorithm with typical inputs. Once the *diff* in Figure 4 is obtained, getting the mapping $\delta_{C \to A}$ is straightforward.

=	0	T	0	T
=	1	G	1	G
=	2	F	2	F
=	3	-	3	-
-	4	β		
+			4	b
+			5	e
+			6	t
+			7	a
=	5		8	
=	6	a	9	a
=	7	c	10	c
=	8	t	11	t
=	9	s	12	s

Figure 4: diff(C, A)

The LCS algorithm works fine when text variations occur only in isolation individually as in Figure 4. Sometimes, however, text variations occur successively, causing what we call the *successive variation* problem. It is illustrated in the *diff* result in Figure 5, where two Unicode characters, 'β' and '–'[4] appear successively. The source position, 5, needs to be precisely mapped to the target position, 8, which however the LCS-diff algorithms cannot find: while the mapping, $\delta(4) \to 4$, is obvious, there is no clue as to which position, among 5, 6, 7 and 8, the next one, $\delta(5)$ to be mapped to.

4 Generalized LCS algorithm

To address the problem of *successive variations*, we need to inform the algorithm of equivalent sequences, e.g., β and *beta*. We call a collection of

[4]long hyphen in Unicode

=	0	T	0	T
=	1	G	1	G
=	2	F	2	F
=	3	-	3	-
-	4	β		
-	5	–		
+			4	b
+			5	e
+			6	t
+			7	a
+			8	-
=	6	i	9	i
=	7	n	10	n
=	8	d	11	d
=	9	u	12	u
=	10	c	13	c
=	11	e	14	e
=	12	d	15	d

Figure 5: The result of LCS-Diff for "*TGF-β–induced*" and "*TGF-beta-induced*"

equivalent sequences a *dictionary*, and modify Algorithm 1 as follows:

```
1:    function GLCS(X[1..m], Y[1..n], D)
        ...
11-1:   a, b := S(X[1..i], Y[1..j], D)
11-2:   if a > 0 then
12:       C[i,j] := C[i-a, j-b] + 1
```

As indicated in line 1, it is invoked with a dictionary, *D*, which is a list of equivalent sequences, e.g., (α, *alpha*), (β, *beta*), and so on. The 11'th line of Algorithm 1, which performs the comparison of the last characters, is modified to perform a general suffix comparison. The suffix comparison function, S, first performs the last-character-comparison for trivial cases, and performs a suffix comparison in variable length when the character comparison fails. The suffix comparison relies on the dictionary, D: if the two strings have matching suffixes in the end according to the dictionary, it returns the length of the suffixes, which is received by *a* and *b* in the modified algorithm.

Following is the modification to Algorithm 2:

```
5-1: a, b := S(X[a..i], Y[b..j])
5-2: if i > 0 and j > 0 and a > 0 then
6-1:   if a = b = 1 and X[i] = Y[i] then
6-2:     push(D, ['=', X[i], Y[j])
6-3:   else
6-4:     for p := i-a+1..i do
6-5:       push(D, ['-', X[p], nil])
6-6:     end for
6-7:     for q := j-b+1..j do
6-8:       push(D, ['+', nil, Y[q])
6-9:     end for
6-10:  end if
7:     i := i - a
8:     j := j - b
```

Using it, the *diff* of the successive variation example is obtained as in Figure 6, where the mapping

=	0	T	0	T
=	1	G	1	G
=	2	F	2	F
=	3	-	3	-
-	4	β		
+			4	b
+			5	e
+			6	t
+			7	a
-	5	–		
+			8	-
=	6	i	9	i
=	7	n	10	n
=	8	d	11	d
=	9	u	12	u
=	10	c	13	c
=	11	e	14	e
=	12	d	15	d

Figure 6: The result of GLCS-Diff for "*TGF-$beta$–induced*" and "*TGF-beta-induced*"

of '–' and '-' is properly represented, solving the problem of successive variations.

As the modified algorithm generalizes the last-character-comparison to the variable-length-suffix-comparison, we call it a *generalized LCS (GLCS)* algorithm. With an empty dictionary, GLCS works exactly the same as LCS. Using a *suffix tree* algorithm, GLCS has the worst case time complexity, $O(mnl)$, where l is the length of the longest entry in the dictionary.

While the performance of GLCS relies on the dictionary, in fact, it works well even with an incomplete dictionary. For example, to get the result in Figure 6, having either (β, *beta*) or (–, -) in the dictionary is enough. This feature contributes to the robustness of GLCS in real world applications.

5 Application and evaluation

The proposed solution is implemented into `PubAnnotation`[5], a storage system for corpora and annotations. The system is developed to share corpora and annotations developed by several annotation projects. The system maintains a collection of texts taken from a number of sources, e.g., PubMed[6] and PubMed Central[7], and supplies them to the annotation projects. The annotations produced by the annotation projects are collected back to `PubAnnotation` for sharing.

As the annotation projects are conducted by different groups independently, when the resulted annotations are submitted to the storage system, the base texts often have been varied from the original, due to, e.g., Unicode-ASCII conversion, tokenization, or accidental insertion or deletion of characters. `PubAnnotation` handles all the mapping and alignment by using the LCS and GLCS algorithms. For performance, LCS is implemented using Hunt-Mcllroy algorithm, and GLCS is implemented as presented in this paper. While using LCS as default, GLCS is only invoked when successive variations are detected. Because successive variations seldom occur, the cost for running GLCS is negligible. Yet, securing a solution for successive variations is important. When experimented with 10 full papers with 54,938 words and 6,007 annotation instances, the text mapping and annotation alignment took less than 10 seconds.

The accuracy of mapping and alignment is thoroughly verified using 10 full text papers with 58,360 words and 7,315 span annotations. Two versions of dictionary for GLCS were prepared: (A) one for all the standard set of Unicode characters[8], and (B) another only for the Unicode characters for whitespace and punctuation symbols. The system successfully aligned all the annotations even with the smaller one, (B). It indicates that when successive variations occur, in most cases, whitespace or punctuation symbols are mixed in it. At least it was the case in our application.

Another 10 full text papers with 54,369 words and 5,729 annotations were used for further verification. While keeping using the dictionary (B), the system is implemented to alert when a unsolvable case is detected. During the processing of the 10 papers, the alert was issued only once, which was caused by the Unicode sequence, $\Delta\Delta$. When the larger dictionary, (A), was used, the problem was not observed. So, it is true that the more complete the dictionary is, the higher the accuracy will be. The empirical results also suggest that, together with the alerting system, the proposed solution works reasonably well, even with minimal size of dictionary.

6 Conclusions

The solution presented in this paper is freely available as a open source Ruby library and also as a free service through the `PubAnnotation` storage system. We expect it to contribute to reduce the cost of community for interoperable use of corpora and annotations.

[5] http://pubannotation.org

[6] http://www.ncbi.nlm.nih.gov/pubmed

[7] http://www.ncbi.nlm.nih.gov/pmc/

[8] As implemented in the standard *unicode* library.

References

L. Bergroth, H. Hakonen, and T. Raita. 2000. A survey of longest common subsequence algorithms. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, SPIRE '00, pages 39–, Washington, DC, USA. IEEE Computer Society.

Razvan Bunescu, Ruifang Ge, Rohit J. Kate, Edward M. Marcotte, Raymond J. Mooney, Arun K. Ramani, and Yuk Wah Wong. 2004. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial Intelligence in Medicine*, 33(2):139–155.

David Campos, Srgio Matos, Ian Lewin, Jos Lus Oliveira, and Dietrich Rebholz-Schuhmann. 2012. Harmonization of gene/protein annotations: towards a gold standard medline. *Bioinformatics*, 28(9):1253–1261.

K. Bretonnel Cohen, Philip V Ogren, Lynne Fox, and Lawrence Hunter. 2005. Empirical data on corpus design and usage in biomedical natural language processing. In *AMIA annual symposium proceedings*, pages 156–160.

Kristofer Franzén, Gunnar Eriksson, Fredrik Olsson, Lars Asker, Per Lidén, and Joakim Cöster. 2002. Protein names and how to find them. *International Journal of Medical Informatics*, 67(13):49 – 61.

James W. Hunt and M. Douglas McIlroy. 1976. An Algorithm for Differential File Comparison. Technical Report 41, Bell Laboratories Computing Science, July.

Nancy Ide and Laurent Romary. 2004. International standard for a linguistic annotation framework. *Nat. Lang. Eng.*, 10(3-4):211–225, September.

Helen Johnson, William Baumgartner, Martin Krallinger, K Bretonnel Cohen, and Lawrence Hunter. 2007. Corpus refactoring: a feasibility study. *Journal of Biomedical Discovery and Collaboration*, 2(1):4.

Jin-Dong Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. 2003. GENIA corpus - a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl. 1):i180–i182.

Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. 2009. Overview of BioNLP'09 Shared Task on Event Extraction. In *Proceedings of Natural Language Processing in Biomedicine (BioNLP) NAACL 2009 Workshop*, pages 1–9.

Dietrich Rebholz-Schuhmann, Antonio Yepes, Chen Li, Senay Kafkas, Ian Lewin, Ning Kang, Peter Corbett, David Milward, Ekaterina Buyko, Elena Beisswanger, Kerstin Hornbostel, Alexandre Kouznetsov, Rene Witte, Jonas Laurila, Christopher Baker, Cheng-Ju Kuo, Simone Clematide, Fabio Rinaldi, Richard Farkas, Gyorgy Mora, Kazuo Hara, Laura I Furlong, Michael Rautschka, Mariana Neves, Alberto Pascual-Montano, Qi Wei, Nigel Collier, Md Chowdhury, Alberto Lavelli, Rafael Berlanga, Roser Morante, Vincent Van Asch, Walter Daelemans, Jose Marina, Erik van Mulligen, Jan Kors, and Udo Hahn. 2011. Assessment of ner solutions against the first and second calbc silver standard corpus. *Journal of Biomedical Semantics*, 2(Suppl 5):S11.

Yue Wang, Jin-Dong Kim, Rune Sætre, Sampo Pyysalo, Tomoko Ohta, and Jun'ichi Tsujii. 2010. Improving the inter-corpora compatibility for protein annotations. *Journal of Bioinformatics and Computational Biology*, 8(5):901–916.