# Multi-level Similar Segment Matching Algorithm for Translation Memories and Example-Based Machine Translation

**Emmanuel PLANAS**
**Cyber Solutions Laboratories**
**2-4, Hikaridai Seika-cho Soraku-gun**
**Kyoto, 619-0237 Japan**
**planas@soy.kecl.ntt.co.jp**

**Osamu FURUSE**
**Cyber Solutions Laboratories**
**2-4, Hikaridai Seika-cho Soraku-gun**
**Kyoto, 619-0237 Japan**
**furuse@soy.kecl.ntt.co.jp**

## Abstract

We propose a dynamic programming algorithm for calculating the similarity between two segments of words of the same language. The similarity is considered as a vector whose coordinates refer to the levels of analysis of the segments. This algorithm is extremely efficient for retrieving the best example in Translation Memory systems. The calculus being constructive, it also gives the correspondences between the words of the two segments. This allows the extension of Translation Memory systems towards Example-based Machine Translation.

## Introduction

In Translation Memory (TM) or Example-Based Machine Translation (EBMT) systems, one of the decisive tasks is to retrieve from the database, the example that best approaches the input sentence. In Planas (1999) we proposed a two-step retrieval procedure, where a rapid and rough index-based search gives a short list of example candidates, and a refined matching selects the best candidates from this list. This procedure drastically improves the reusability rate of selected examples to 97% at worst, for our English-Japanese TM prototype; with the classical TM strategy, this rate would constantly decline with the number of non matched words. It also allows a better recall rate when searching for very similar examples.

We describe here the Multi-level Similar Segment Matching (MSSM) algorithm on which is based the second step of the above retrieval procedure. This algorithm does not only give the distance between the input and the example source segments, but also indicates which words would match together. It uses F different levels of data (surface words, lemmas, parts of speech (POS), etc.) in a combined and uniform way. The computation of the worst case requires F*m*(n-m+2) operations, where m and n are respectively the lengths of the input and the candidate (m<=n). This leads to a linear behavior when m and n have similar lengths, which is often the case for TM segments[1]. Furthermore, because this algorithm gives the exact matching links (along with the level of match) between all of the words of the input and the candidate sentence, it prepares the transfer stage of an evolution of TM that we call Shallow Translation. This involves substituting in the corresponding translated candidate (stored in the memory), the translation of the substituted words, provided that the input and the candidate are "similar enough".

## 1 Matching Principle

### 1.1 The TELA Structure

The purpose of this algorithm is to match two segments of words: input I and candidate C. These can each be any sequence of words: phrases, sentences, or paragraphs, for example. Let us consider input I of length m, not as a single segment of surface words, but rather as a group of F parallel layered segments $I^f_{(1<=f<=F)}$ each bearing m tokens. Such a structure is shown in Figure 1, and we call it a TELA structure[2]. On each layer f, the i-th token corresponds to one of the paradigms of the i-th word of input I. In our implementation, we use a shallow analyzer that gives three paradigms (F=3) for each surface

[1] We use this algorithm on a sorted list of already similar sentences, retrieved with the help of an index.

[2] The idea of this structure is already in Lafourcade's LEAF (1993), and is explained in Planas (1998).

| **C** | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $C_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $C^1$ | ~~Nikkei~~ | ~~Journal~~ | ~~reported~~ | ~~that~~ | ~~NTT~~ | ~~really~~ | **stayed** | ~~strong~~ | ~~Monday~~ |
| $C^2$ | ~~nikkei~~ | ~~journal~~ | ~~report~~ | ~~that~~ | ~~NTT~~ | ~~really~~ | **stay** | **strong** | ~~Monday~~ |
| $C^3$ | **PN** | **noun** | **verb** | ~~conj~~ | **PN** | ~~adv~~ | **verb** | **adj** | **noun** |

**Matching zone**

| **I** | $I_1$ | $I_2$ | $I_3$ | $I_4$ |
|---|---|---|---|---|
| $I^1$ | ~~Sony~~ | **stayed** | ~~stronger~~ | ~~Tuesday~~ |
| $I^2$ | ~~Sony~~ | **stay** | **strong** | ~~Tuesday~~ |
| $I^3$ | **PN** | **verb** | **adj** | **noun** |

*Figure 1: Example of matching TELA structures*

word of the segments: the surface word itself (f=1), its lemma (f=2), and its POS tag (f=3). Because we do not need a syntactic analyzer, the time required for this analysis is not an handicap, moreover such parsers are available for many languages. Let C be a candidate segment of length n, for matching input I of length m (n>=m). The basic problem involves matching the elements of the set $(C^f_i)_{f<=F,\ i<=n}$ to those of $(I^f_j)_{f<=F,\ j<=m}$. Only three layers are shown in the following examples but other types of layers, like semantics, or even non linguistic information like layout features can be considered, as in Planas (1998). Our algorithm is written for the general case (F layers).

## 1.2 Edit Distance based Similarity

We consider a match from C to I as an edit distance process. This edition uses a sequence of basic edit operations between the words of the segments, like in Wagner & Fisher (1974) who used four basic operations: deletion, insertion, strict and equal substitution between the letters of a word. This approach has also been followed by Gale & Church (1993) for their alignment algorithm, with six operations. Here, we only consider **deletions and equalities (i.e. equal substitutions)**: F+1 basic operations in total[3]. One equality corresponds to each of the F layers, and a deletion affects all layers at once. In Figure 1, the items in bold match each other, and the strikethrough ones have to be deleted. The edition of C into I involves five deletions ("Nikkei", "journal", "reported", "that", "really"), one equality at layer 1 ("stayed"), two at layer 2 ("stay", "strong"), and four at layer 3 ("PN", "verb", "adj", "noun"). At the Word level, the similarity between the two segments is considered to be the relative number of words of the input segment that are matched by some word of the candidate segment in the matching zone (from "NTT" to "Monday" in our example): 1/4 in Figure 1. The same similarity can be considered at different levels. Here, the lemma similarity is 2/4, and the POS similarity is 4/4. We consider the total similarity as a vector involving all layer equalities, plus deletions: $\sigma(C, I) = (1/4, 2/4, 4/4, 1-1/4, 1-5/9)$

The fourth coordinate counts the complementary proportion of deletions in the "matching zone" of the candidate C. The last coordinate counts the same proportion, relatively to the whole candidate. We take the complement to 1 because, the more deletions there are, the smaller the similarity becomes.

When different Ci candidates are possible for matching I, the greatest $\sigma(Ci_0, I)$, according to common the partial order on vectors, determines the best candidate $Ci_0$.

## 1.3 Matching Strategy

### 1.3.1 Basics

We try to match each word $C_i$ of candidate C, to a word $I_j$ of input I. $C_i$ matches $I_j$ if one of the paradigms of $C_i$ equals one of the paradigms of $I_j$ at the same level f, i.e. if $C^f_i$ and $I^f_j$, are equal. When a failure to match two words with their paradigms $C^f_i$ to $I^f_j$ occurs at a given level f, we try to match the words at the next upper level f+1: $C^{f+1}_i$ and $I^{f+1}_j$. When all of the possible layers of the two words have been tried without success, we try to match the next word $C_{i+1}$ to the same $I_j$. If $C_i$ does not match any word of I at any

[3] Lepage (1998) also uses deletions and one level of equality for calculating his "pseudo-distance", for getting the similarity between two strings.

level, we consider that it has to be deleted. All words of I have to be matched by some word of C: no insertion is allowed (see section 1.3.4).

### *1.3.2 Lazy match*

With TM tools, if some useful candidates are found, they usually utilize words similar to the input words because translation memories are applied within very similar documents, most of the time between ancient and newer versions of a same document. When the priority is rapidity (rather than non-ambiguity), we can consider that a match is reached as soon as a word of C and a word of I match at a certain layer f. It is not necessary to look at upper levels, for they *should* match because of the expected similarity between the input and the candidate. The previous example illustrates this. As upper levels are not tested, this allows a gain in the number of iterations of the algorithm. Experiments (see Planas (1999)) have confirmed this to be a correct strategy for TM. That's why, we consider from now on dealing with such a lazy match.

### *1.3.3 Exhaustive match*

In the most general case, ambiguity problems prevent us from employing the lazy strategy, and a correct match requires that whenever two items $C^f_i$ and $I^f_j$ match at a certain level f, they should match at upper levels. Here is an example:

| | | | | |
|---|---|---|---|---|
| $C2^1$ | **Sony** | ~~stay~~ | ~~ended~~ | ~~Monday~~ |
| $C2^2$ | **Sony** | **stay** | ~~ended~~ | ~~Monday~~ |
| $C2^3$ | **PN** | ~~noun~~ | ~~verb~~ | **noun** |

| | | | | |
|---|---|---|---|---|
| $I^1$ | **Sony** | ~~stayed~~ | ~~stronger~~ | ~~Tuesday~~ |
| $I^2$ | **Sony** | **stay** | ~~strong~~ | ~~Tuesday~~ |
| $I^3$ | **PN** | ~~verb~~ | ~~adj~~ | **noun** |

*Figure 2: Lemma ambiguity*

In C2, the lemma "stay" of surface word "stay" matches the lemma "stay" of surface word "stayed" of I, but they do not match at the POS level (noun and verb). The algorithm should go to this level to find that there is no match. Once again, however, because this algorithm has been built for TM systems, such ambiguities hardly occur.

### *1.3.4 Insertion*

If some items in I are not matched by any item of C, the match involves an insertion.

***Case of Translation Memories***

If the candidate sentences are to be used by a human translator, s/he will be able to insert the missing word at the right place. Accordingly, a match with insertion can be used for pure TM.

***Case of Shallow Translation (EBMT)***

In the EBMT system we are targeting, we plan to use the matching sub-string of C for adaptation to I *without syntactic rules.* Accordingly, we consider that we do not know where to insert the non matching item: in this case, we force the algorithm to stop if an insertion is needed for matching C and I. From now on, we will follow this position.

### *1.3.5 Trace*

We want the output of the algorithm as a list of triplets $(C^f_i\ I^f_j\ op)_{1<=i<=n}$ called a "trace", where $C^f_i$ corresponds to $I^f_j$ through the "op" operation. We note op="f" an equality at level f, and op="0" a deletion. For Example 1, the trace should be:
(1 0 0) (2 0 0) (3 0 0) (4 0 0) (5 1 3) (6 0 0) (7 2 1) (8 3 2) (9 4 3)

## 2 Adapting Wagner & Fischer, and Sellers algorithms

### 2.1 Algorithm Principle

The Wagner & Fischer (W&F) dynamic programming algorithm in Figure 3 gives the edit distance between C and I:

```
For j=0 to m
    d[j, 0]=j //initiating the columns
For i=1 to n
    d[0, i]=i //initiating the rows
For i=1 to n
    For j=1 to m
        If(I[j]=C[i]) {d=d[i-1, j-1]}//equality
        Else {d=d[i-1, j-1]+1}      // subst.
        d[j,i]=min(d[i-1, j]+1, d[i, j-1]+1, d)
    End For
End For
Print d[n, m]
```

*Figure 3: The Wagner & Fisher algorithm*

The distance is obtained in m*n operations, by building an [m+1, n+1] array (see Figure 6). In addition, W&F (1974) proposed a backtracking procedure, shown in Figure 4, that scans back this array to give a "trace" of the match between

C and I (i.e. it prints the position of the matching words), in (m+n) operations. The trace is then obtained in (mn+m+n) operations in total. This algorithm was previously used in Planas (1998) at each layer of a TELA structure to give a trace by layer. The data from the traces of the different layers were combined *afterwards* for the purposes of TM and EBMT. However, this procedure is not optimal for at least two reasons. First, the layers are compared in an independent way, leading to a waste of time in the case of TM, because the lazy match phenomenon is not used. Second, the combination of the results was processed after the algorithm, and this required a supplementary process. One can imagine that processing the whole data in the flow of the instructions of the algorithm is more efficient.

```
i= $i_0$; j = m;
while (i > 0) and (j > 0)
//del//   if (d[i, j]= d[i-1, j]+ 1) {i = i -1}
//ins//   else if (d[i, j]= d[i, j-1]+ 1) {j = j -1}
          else //equality or substitution
                    print (i, j)
                    i = i - 1; j = j - 1
          end if
end while
```

*Figure 4: W&F backtracking algorithm*

## 2.2 Two operation based minimization

If we look back at the W&F algorithm, shown in Figure 3, the part in bold represents the code involved in the calculus of the next local distance d[i, j]. It testes which of the four basic edit operations (deletion, insertion, equal or strict substitution) gives the lowest partial distance. Nevertheless, we have shown in section 1.3.4 that only deletions and equalities do interest us. We therefore reduce the test in the algorithm to that shown in Figure 5. Furthermore, we initiate the columns of the array with infinite values (huge values in practice) to show that initial insertions are not possible, and the rows to "0", to count the deletions relatively to input I. See Sellers (1980) for a due explanation.

```
If(I[j]=C[i]) {d=d[i-1, j-1]} //equal: no cost
Else {d=inf} //big integer, in theory infinite
d[j,i] = min (d[i-1, j]+1, d) //deletion or equal ?
```

*Figure 5: Matching with deletions and equalities*

An example of the successive scores calculated with this algorithm are shown in Figure 6. The total distance (equal to 1) between C and I appears in the lowest right cell.

The fact that only two operations are used *eradicates the ambiguity* that appears in selecting the next cell in the W&F algorithm backtracking procedure with four operations. In our algorithm, either there is an equality (cost 0), or a deletion (cost 1). The possibility of having the same cost 1 for insertions, deletions, or strict substitutions has been eliminated.

## 2.3 Introducing one equality per level

As mentioned previously, we need to match items at different layers. We introduce here two new points to deal with this:

- In order to keep the score for each equality deletion, d[i,j] is a *vector* instead of a number: d[i,j]=[$\text{score}_1$,...,$\text{score}_F$, $\text{score}_{--}$].
- In this vector, $\text{score}_1$ through $\text{score}_F$ store the *number of equalities* for each layer f, and $\text{score}_{--}$ records the number of deletions, as in W&F (underlined in the arrays).

| | | $I_j$ | $I^1{}_1$ | $I^1{}_2$ | $I^1{}_3$ | $I^1{}_4$ | $I^1{}_5$ |
|---|---|---|---|---|---|---|---|
| | | | *First* | *press* | *the* | *red* | *button* |
| $C^1{}_i$ | | *0* | *inf* | *inf* | *inf* | *inf* | *inf* |
| $C^1{}_1$ | *First* | *0* | **0** | inf | inf | inf | inf |
| $C^1{}_2$ | *press* | *0* | 1 | **0** | inf | inf | inf |
| $C^1{}_3$ | *only* | *0* | 2 | **1** | inf | inf | inf |
| $C^1{}_4$ | *the* | *0* | 3 | 2 | **1** | inf | inf |
| $C^1{}_5$ | *red* | *0* | 4 | 3 | 2 | **1** | inf |
| $C^1{}_6$ | *button* | *0* | 5 | 4 | 3 | 2 | $\boxed{\mathbf{1}}$ |

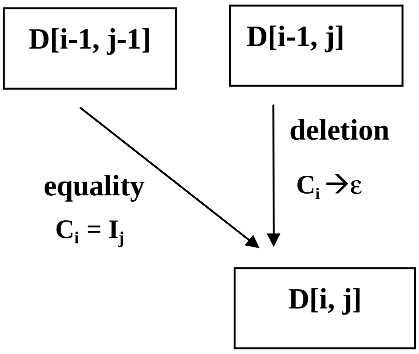

*Figure 6: Successive scores produced by the adapted W&F algorithm*

|  |  |  | $I^f_j$ | 0 | $I_1$ | $I_2$ | $I_3$ | $I_4$ |
|---|---|---|---|---|---|---|---|---|
|  |  |  | $I^1$ | *word* | Sony | stays | strong | Tuesday |
|  |  |  | $I^2$ | *lem* | Sony | stay | strong | Tuesday |
| $C^f i$ | $C^1$ | $C^2$ | $C^3/I^3$ | *POS* | PN | verb | adj | PN |
| *0* | *word* | *lem* | *POS* | *0000* | *000inf* | *000inf* | *000inf* | *000inf* |
| $C_1$ | Sony | Sony | PN | *0000* | **1000** | 000inf | 000inf | 000inf |
| $C_2$ | reported | say | verb | *0000* | 1001 | **1010** | 000inf | 000inf |
| $C_3$ | that | that | conj | *0000* | 1002 | 1011 | 000inf | 000inf |
| $C_4$ | NTT | NTT | PN | *0000* | **0010** | 1012 | 000inf | 000inf |
| $C_5$ | stayed | stay | Verb | *0000* | 0011 | **0110** | 000inf | 000inf |
| $C_6$ | stronger | strong | Adj | *0000* | 0012 | 0111 | **0210** | 000inf |
| $C_7$ | Tuesday | Tuesday | PN | *0000* | 0010 | 0112 | 0211 | **1210** |
| $C_8$ | morning | morning | noun | *0000* | 0011 | 0113 | 0212 | **1211** |

*Figure 7: Introducing a vector of deletion and layer equalities scores*

Figure 7 shows an example of different score vectors involved in a match. To calculate the successive d[i,j], we use the algorithm of Figure 5 adapted for F levels in Figure 8.

```
If($I^f[j]=C^f[i]$)
    $d_e=[d^1[i-1,j-1], …,d^F[i-1,j-1]+1,d^{-}[i-1,j-1]\ ]$
Else
    $d_e=[0,…,0,inf]$
End If
$d_d=[d^1[i-1,j], …,d^f[i-1,j], .. ,d^F[i-1,j], d^{-}[i-1,j]+1]$
$d[j,i] = max\ (d_e, d_d)$        // equality or deletion
```

*Figure 8: Adapting the algorithm to F levels*

We first try to get the maximum number of equalities and then the minimum of deletions. Each time we find a new match in the first column, we start a new path (see $I^1$ matching with $C^1$, $C^4$ and $C^7$ in Figure 7). If one of the vectors of the last column of the array is such that: $SUM_{1<=f<=F}$ ($score_f$) = m, there is a matching substring of C in which there is a matching word for each of the words of I: this constitutes a solution. In our example, cell (7, 4), with score 1210 shows that there is a sub chain of the candidate that matches the input with 1, 2, and 1 matches at the word, lemma, and POS levels and 0 deletions. Cell (8, 4) indicates a similar match, but with 1 deletion ("morning"). The best path then ends at cell (7,4). Starting from this cell, we can retrieve the full solution using the W&F backtrack algorithm adapted to F levels.

This approach allows us to choose as compact a string as possible. When there are several possible paths, like in Figure 9, the algorithm is able to choose the best matching sub-string. If we are looking for a similarity involving first

|  |  |  | $I^f_j$ | 0 | $I_1$ | $I_2$ | $I_3$ | $I_4$ |
|---|---|---|---|---|---|---|---|---|
|  |  |  | $I^1$ | *word* | Sony | stays | strong | Tuesday |
|  |  |  | $I^2$ | *lem* | Sony | stay | strong | Tuesday |
| $C^f i$ | $C^1$ | $C^2$ | $C^3/I^3$ | *POS* | PN | verb | adj | PN |
| 0 | *word* | *lem* | *POS* | *0000* | *000inf* | *000inf* | *000inf* | *000inf* |
| $C_1$ | Sony | Sony | PN | *0000* | **1000** | 000inf | 000inf | 000inf |
| $C_2$ | stayed | say | verb | *0000* | 1001 | **1100** | 000inf | 000inf |
| $C_3$ | stronger | strong | adj | *0000* | 1002 | 1101 | **1200** | 000inf |
| $C_4$ | Tuesday | Tuesday | PN | *0000* | 1000 | 1102 | 1201 | **2200** |
| $C_5$ | and | and | conj | *0000* | 1001 | 1103 | 1202 | **2201** |
| $C_6$ | NTT | NTT | PN | *0000* | **0010** | 1104 | 1203 | 2202 |
| $C_7$ | stayed | stay | Verb | *0000* | 0011 | **0110** | 1204 | 2203 |
| $C_8$ | stronger | strong | Adj | *0000* | 0012 | 0111 | **0210** | 2204 |
| $C_9$ | Tuesday | Tuesday | PN | *0000* | 0010 | 0112 | 0211 | **1210** |
| $C_{10}$ | morning | morning | noun | *0000* | 0011 | 0113 | 0212 | **1211** |

*Figure 9: Selecting the best concurrent sub segment*

surface word matches, then lemmas and parts of speech, then cell (4,4) of score 2200 will be chosen. This strategy can be adapted to particular needs: it suffices to change the order of the scores in the vectors.

## 3 Optimizing

### 3.1 Triangularization of the array

In this algorithm, for each $I_j$, there must be at least one possible matching $C_i$. Hence, in a valid path, there are at least m matches. As a match between $C_i$ and $I_j$ occurs when "stepping across a diagonal", the (m-1) first diagonals (from the lower left corner of the array) can not give birth to a valid path. Therefore, we do not calculate d[i,j] across these small diagonals. Symmetrically, the small diagonals after the last full one (in the upper right corner) cannot give birth to a valid path. We then also eliminate these (m-1) last diagonals. This gives a reduced matrix as shown in the new example in Figure 10. The computed cells are then situated in a parallelogram of dimensions (n-m+1) and m. The results is: only m(n-m+1) cells have to be computed. Instead of initiating the first row 0 to "inf", we initiate the cells of the diagonal just before the last full top diagonal (between cell (0,1) and cell (3,4) in Figure 10) to "000inf" to be sure that no insertion is possible.

### 3.2. Complexity

The worst time complexity of this algorithm is F-proportional to the number of cells in the computed array, which is m*(n-m+1). With the "lazy" strategy, all F levels are often not visited. As the number of cells computed by the W&F algorithm is m*n, our algorithm is always more rapid. The backtracking algorithm takes m+n operations in the W&F algorithm, as well as in our algorithm, leading to m(n-m+2)+n operations in the MSSM algorithm, and m(n+1)+n operations in the W&F algorithm**.** The general complexity is then sub-quadratic. When the lengths of both segments to be compared are similar (like it often happens in TMs), the complexity tends towards linearity.

The two graphics in Figure 11 show two interesting particular cases (m=n and m running from 1 to n=10), comparing W&F and our algorithm. For strings of similar lengths, the longer they are, the more the MSSM algorithm becomes interesting. When n is fixed, the MSSM algorithm is more interesting for extreme values of the length of I: small and similar to n.

## Conclusions

The first contribution of this algorithm is to provide TM and EBMT systems with a precise and quick way to compare segments of words with a similarity vector. This leads to an almost complete eradication of noise for the matter of retrieving similar sentences in TM systems (97% "reusability" in our prototype). The second is to offer an unambiguous word to word matching through the "trace". This last point opens the way to the Shallow Translation paradigm.

| | | | $I^f_j$ | 0 | $I_1$ | $I_2$ | $I_3$ | $I_4$ |
|---|---|---|---|---|---|---|---|---|
| | | | $I^1$ | *word* | Sony | stays | strong | Tuesday |
| | | | $I^2$ | *lem* | Sony | stay | strong | Tuesday |
| $C^f i$ | $C^1$ | $C^2$ | $C^3/I^3$ | *POS* | PN | verb | adj | PN |
| 0 | *word* | *lem* | *POS* | *0000* | *000inf* | | | |
| $C_1$ | Sony | Sony | PN | *0000* | **1000** | *000inf* | | |
| $C_2$ | reported | say | verb | *0000* | 1001 | **1010** | *000inf* | |
| $C_3$ | that | that | conj | *0000* | 1002 | 1011 | 000inf | *000inf* |
| $C_4$ | NTT | NTT | PN | *0000* | **0010** | 1012 | 000inf | 000inf |
| $C_5$ | stayed | stay | Verb | *0000* | 0011 | **0110** | 000inf | 000inf |
| $C_6$ | stronger | strong | Adj | *0000* | | 0111 | **0210** | 000inf |
| $C_7$ | Tuesday | Tuesday | P noun | *0000* | | | 0211 | **1210** |
| $C_8$ | morning | morning | noun | *0000* | | | | **1211** |

*Figure 10: Eliminating left and right small diagonals*

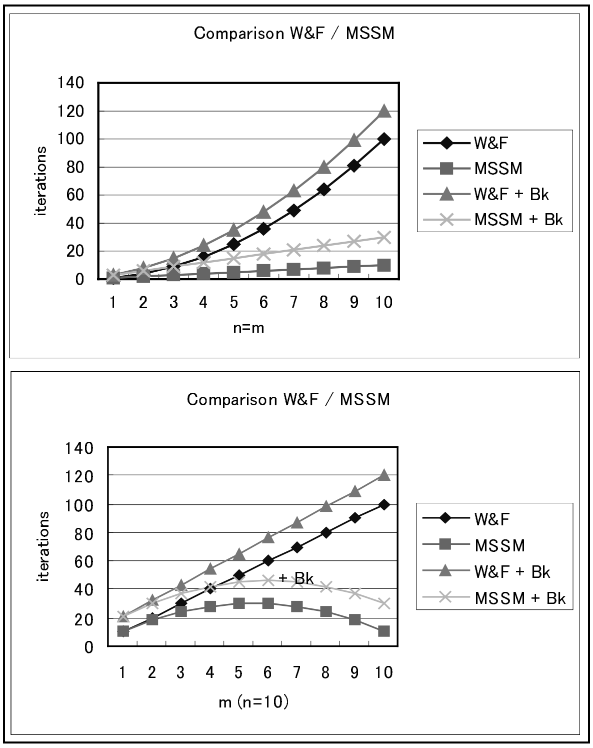

*Figure 11: Comparing the Wagner & Fisher and MSSM algorithms*

For more information about the use of this algorithm, please refer to Planas (1999). These two contributions bring in the main difference with relative research[4] concentrating on similarity only, represented by a sole integer.

The TELA structure, that allows the parallel use of different layers of analysis (linguistic paradigms, but possibly non linguistic information) is essential to this work because it provides the algorithm with the supplementary information classical systems lack.

The fact that the shallow parser (lemmas, POS) is ambiguous or not does not affect significantly the performance of the algorithm. If the same parser is used for both example and input segments, parallel errors compensate each other. Of course, these errors do have an influence for EBMT: the non ambiguity is then a must.

A first evaluation of the MSSM speed gives 0.5 to 2 milliseconds for comparing only[5] two randomly chosen English or Japanese sentences over 3 levels (word, lemmas, POS). The implementation has been done with a DELL Optiplex GX1 233 Mhz, Window NT, Java 118.

This algorithm can be improved in different ways. For speed, we can introduce a similarity threshold so as not to evaluate the last cells of the columns of the computed array as soon as the threshold is overtaken. For adaptability, being able to deal with a different number of tokens according to each layer will allow us to deal nicely with compound words.

In short, if the basis of this matching algorithm is the W&F algorithm, other algorithms can be adapted similarly to deal with multi-level data.

## Acknowledgements

Thanks to Takayuki Adachi, Francis Bond, Timothy Balwin, and Christian Boitet for their useful remarks and fruitful discussions.

## References

Cranias, L., Papageorgiou, H., & Piperidis, S. (1997) *Example retrieval from a Translation Memory.* Natural Language Engineering 3(4), Cambridge University Press, pp. 255-277.

Gale, W.A. & Church, K.W. (1993) *A program for Aligning Sentences in Bilingual Corpora*. Computational Linguistics, ACL, Vol. 19, No. 1.

Lafourcade M. (1993) *LEAF, ou comment garder l'Originalité de l'ambiguité*. Actualité Scientifique - Troisièmes Journées Scientifiques Traductique-TA-TAO, Montréal, Canada, AUPELF-UREF, Vol. 1/1, pp. 165-185.

Lepage Y. (1998) *Solving analogies on words: an algorithm*. Coling-ACL'98, Vol. I, pp. 728-734.

Sellers, P.H. (1980) *The theory and computation of evolutionary distances: pattern recognition*. Journal of Algorithms, Vol. 127, pp. 359-373.

Thompson Henry S. & Brew Chris (1996) *Automatic Evaluation of Computer Generated text: Final Report on the TextEval Project.* Human Communication Research Center, University of Edinburg.

Wagner, A. R. & Fischer M. (1974) *The String-to-String Correction Problem*. Journal of the ACM, Vol. 21, #1, pp. 168-173.

Planas, E. (1998) *TELA: Structures and Algorithms for Memory-Based Machine Translation*. Ph.D. thesis, University Joseph Fourier, Grenoble.

Planas, E. & Furuse O. (1999) *Formalizing Translation Memories.* Machine Translation Summit VII, Singapore, pp. 331-339

[4] Cranias et al. (1997), Thompson & Brew (1994), or in a more specific way, Lepage (1998)

[5] Without the shallow analysis