Language and Computation

LING 227 01 / 627 01 / PSYC 327 01

***Minimal Edit Distance* is a distance metric**

Remarks on Assignment #, Problem 3

1 The problem

Your assignment (`http://birot.hu/courses/2014-LC/LC-assignment-2.pdf`) was to demonstrate that *Minimal Edit Distance* (or, shortly, *Edit Distance*, also known as *Levenshtein Distance*) is a *distance metric*, which satisfies the following conditions:

1. $D(a, b) \geq 0$ (*non-negativity*)
2. $D(a, b) = D(b, a)$ (*symmetry*)
3. $D(a, b) = 0$ if and only if $a = b$ (*coincidence axiom*)
4. $D(a, b) + D(b, c) \geq D(a, c)$ (*subadditivity*, or *triangle inequality*)

Please note that the conjunction "if and only if" in the *coincidence axiom* (a.k.a. *identity of indiscernibles*) subsumes two statements, which are most often proven separately: (1) if $a = b$, then $D(a, b) = 0$, and (2) if $D(a, b) = 0$, then $a = b$.

2 Introductory remarks

2.1 Goals of the assignment

The goal of this assignment was manifold: to have you practice the concept of *Minimal Edit Distance* and its algorithmic computation, to train your formal and mathematical skills in general, as well as to show you how a certain topic can be tackled from different angles, on different levels. These were the reasons why the text of the assignment included the sentence "in order to receive the maximum amount of points, you should base at least some parts of the proof on the algorithm: explain why the algorithm returns a value that satisfies the criteria of a distance metric."

From a formal perspective, a perfect proof based on the most formal mathematical definitions of the concepts is the best possible solution; but not from a pedagogical perspective. This is why I valued higher the solutions demonstrating that their authors are able to use a broader scope of approaches.

The motivation behind this current note is to help you reach these goals.

2.2 On the “reasonable conditions”

Most of the students correctly understood that the remark on “reasonable conditions” referred to requirements such as the costs should be positive, symmetric, etc., as discussed in a moment, without which the above criteria for being a distance metric are not (necessarily/always) satisfied. Some of you also demonstrated that these conditions are not only *sufficient*, but also *necessary* for MED to be a distance metric. In any case, one should always start a discussion by *explicitly* enumerating the assumptions to be deployed in the proofs: those of you who did not do so received less points for a less clear train-of-thought.

As always in mathematics, one should try to provide general results: conditions and statements should be made as broad as possible, so that what you prove can be employed in many different conditions. Many of you restricted themselves to unity insertion costs, deletion costs and substitution costs: even though these are indeed “default values”, and the standard values used to illustrate the method, they are by far not the only cost values used in various applications. Therefore, I appreciated more the solutions that did not rely on such an unnecessarily restricting assumption.

Moreover, some people—probably by misunderstanding a remark in the textbook—supposed unity costs for insertion and deletion, and a double cost for substitution. Such a case (and even the case when substitution costs are higher than the sum of the insertion and deletion costs) are theoretically possible, allowed by the proofs below, but impractical: it practically eliminates the substitution operation, because the latter can be replaced by the not more expensive series of a deletion and an insertion. Excluding substitution may be useful in some applications, but less useful in other ones: therefore, our proofs below do not make any assumption on how the summed up costs of an insertion and a deletion relate to the substitution cost.

As a side note: our approach does not include a *transposition* operation (metathesis), although it would be also useful. In theory, re-arranging neighboring characters can be replaced by the deletion of one of the two letters, followed by an insertion of the same letter at a different place. In fact, metathesis is a frequent operation in the grammars (morphologies) of languages, in their history, but also in typing errors. In turn, many systems could be improved if the edit distance metric were enriched by this operation—which makes only sense if the cost of metathesis is less than the sum of an insertion and a deletion. Moreover, cheap transposition could be restricted to adjacent or near characters, distinguishing it from costly deletion and remote re-insertion.

Finally, as mentioned in class and as featured on the pseudo-code of the MED algorithm, different characters may have different insertion, deletion and substitution costs. For example, an old-day copyist might very simply have substituted the letter `O` for the digit `0`, and the deletion of certain sounds are much more likely in the history of languages than the deletion of other sounds. It follows that our argumentation below should also allow for this possibility.

2.3 The way to the solution is not a solution

Lecturers and scholarly articles often introduce concepts, proofs, algorithms, etc., by guiding their audience through a train of thought, pretending as if the original discovery were made via this stream. Believe me, in most cases it was not! However, this kind of rhetoric might make you believe that you also should or could present your results the way they originally emerged.

After you have solved a problem, you should take a step back—ideally, wait a day or two—and only then write your results down. When you commit your results to writing, you should most probably reorganize it, and present it using a different logic. To some extent, this is common sense, and yet, a surprising fact was the proliferation of *proofs by contradiction* among your solutions.

People (including myself) tend to prove mathematical propositions by assuming the opposite. This phenomenon might be due to the way we think: we can better understand what a proposition means if we toy with the opposite idea; if the opposite turns out to be impossible, we better understand why the current proposition must be true. However, proofs by contradiction are very often not very elegant, and sometimes even vague (at least, it turned out to be so among your solutions). Therefore, whenever you come up with a proof by contradiction, you should try to revise it: can you not reverse the argument, and present a more elegant, more convincing and clearer positive proof?

3 Assumptions

Let $c_i(x)$, $c_d(x)$ and $c_s(x, y)$ be the *insertion cost* of character x, the *deletion cost* of character x, and the *substitution cost* of replacing character x with character y, respectively. Note that (in opposition to what some of you wrote) x and y are always characters (elements of Σ), and never strings (elements of Σ^*; hence, not even the empty string).

We will make use of the following assumptions:

1. For all $x \in \Sigma$, $c_i(x) > 0$ and $c_d(x) > 0$. (The cost of either inserting or deleting a character is positive.)

2. For all x and $y \in \Sigma$, $c_s(x, y) \geq 0$. (Substitution costs are non-negative.)

3. For all x and $y \in \Sigma$, $c_s(x, y) = 0$ if and only if $x = y$. (Substitution costs are positive, with the exception of replacing a character with itself. The cost of the latter operation must be null.)

4. For all $x \in \Sigma$, $c_i(x) = c_d(x)$. (The cost of inserting a character is equal to the cost of deleting it.)

Note that we do not assume that all insertion costs (deletion costs) are equal. We do not assume either that $c_s(x, y) \leq c_d(x) + c_i(y)$: if this latter inequality does not hold, then although it is still possible to employ the substitution operation, it will not be used by minimal-cost alignments.

You would come up with these assumptions not *before*, but *while* thinking about the proofs. But after you would solve the problem, and *before* you start writing your solutions down, you should review them.

The clarity and convincing power of your proofs will highly benefit from listing your assumptions *before* you enter the details of the proofs. Thus, the conditions of your general result ("minimal edit distance is a distance metric") becomes clearer to the reader (and to yourself). Unfortunately, many of you only mentioned these assumptions (explicitly, or, even worse, implicitly) at random points of the proofs, obscuring hereby the validity of your results.

4 Informal proofs

Jurafsky and Martin introduces Minimal Edit Distance informally using two different perspectives. Let me refer to the first one as the *alignment perspective*, and to the second one as the *transformation path perspective.* The two are equivalent in practice, but not in theory, as we shall soon see it.

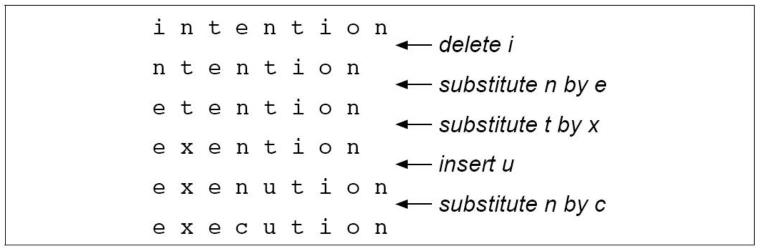
The first, two-level-only approach aligns the two strings. Characters and empty strings in the first string are made correspond to characters and empty strings in the second string (here an asterisk stands for the empty string):

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Each correspondence has a cost: $c_s(x, y)$ if the upper string contains x and the lower string contains y, $c_i(x)$ if the upper string contains the empty string and the lower string contains x, and finally $c_d(x)$ if the upper string contains x and the lower string contains the empty string in this correspondence. (Empty string to empty string correspondences will not be allowed or will be assigned a zero cost by definition.)

It is this alignment approach which is taken by the Minimal Edit Distance Algorithm. It does not allow for a character to be inserted *and* then deleted while we "go" from the first string to the second string. We automatically assume that such an operation pair is unnecessary, does not improve the overall costs, and so we even do not consider it.

A different approach to Minimal Edit Distance is taken by the formal definition I provided as the appendix of the assignment. It considers a series of elementary *transformations* (insertion, deletion and substitution), each with a cost. These operations define a "path" of gradually changing strings. Minimal Edit Distance is the sum of the costs of these single edit operations not performed in parallel but in a sequence, along the *path* connecting the upper string to the lower string:



When discussing the informal proofs below, it might be more useful to have this second approach in mind. Still, you can change the word "path" to the word "alignment" below, if you prefer the first approach.

4.1 Non-negativity: $D(x,y) \geq 0$

In order to get from x to y, we have to perform zero, one or more operations. Performing no operation has a zero cost. From the assumptions above it follows that any operation incurs a non-negative cost. Hence, a single operation path from x to y has a non-negative cost. If a path involves more operations, then its cost is the sum of a number of non-negative costs, which is again a non-negative number. The minimal edit distance path is therefore chosen from a set of paths each with non-negative cost, and so it will also have a non-negative cost.

4.2 Symmetry: $D(x,y) = D(y,x)$

First, we show that by the assumptions made above, reversing a path (that is, reversing the upper and the lower string) does not alter the cost of that specific path (alignment). Indeed, when we reverse the path (when we reverse the upper and the lower string), insertions become deletions, deletions become insertions, and x-to-y substitutions become y-to-x substitutions. If $c_i(x) = c_d(x)$ and $c_s(x,y) = c_s(y,x)$ for all x and y, then the same costs are summed up along both paths, even if in the opposite order.

Second, let us consider the minimal edit path (alignment) from x to y. Reversing this path (alignment), we obtain a path (alignment) from y to x that has the same cost. It remains to show that this is the minimum edit distance path (alignment) from y to x. We can show this by contradiction: if there were a lesser edit distance path from y to x, then the reverse of this path would be a path from x to y shorter than the one we have just supposed to be minimal.

We can rephrase this proof in a more elegant way, without a proof by contradiction. Let p be the minimal edit distance path (alignment) from x to y, and p^r the reverse path (alignment) from y to x. The fact that p is minimal means that for any path (alignment) p' from x to y, the cost of p' is greater than or equal to the cost of p. Now we show that the cost of p^r is minimal among the costs of the paths from y to x. Let q be some path (alignment) from y to x. Then, the reverse of q is a path (alignment) from x to y, and therefore its cost is greater than or equal to the cost of p. Since the costs of p and of p^r are equal, and the costs of r and of its reverse are also equal, it follows that p^r is the minimal edit alignment between from y to x.

As $D(x, y)$ is the cost of p, and $D(y, x)$ is the cost of p^r, it follows that $D(x, y) = D(y, x)$.

4.3 Coincidence axiom: $D(x, y) = 0$ iff $x = y$

First, we show that if $x = y$, then $D(x, y) = 0$. Namely, if $x = y$, then no operation is needed to get from x to y, and so there is a path (alignment) with zero cost: $D(x, y)$ is at most 0. But by non-negativity, the distance cannot be less than zero. Consequently, $D(x, y) = 0$.

Second, suppose that $D(x, y) = 0$, which means that there is a zero-cost path (alignment) from x to y. All operations have non-negative costs by the assumptions above, from which it follow that the minimal edit distance path (alignment) from x to y cannot contain positive cost operations. The minimal edit distance path (alignment) either contains no operation at all, or only contains zero-cost operations. In the former case, $x = y$. In the latter case, observe that the assumptions allow for one type of zero-cost operation only: substitution to itself. Such operations make no change to the string; hence, $x = y$ again.

4.4 Triangle inequality: $D(x, y) + D(y, x) \geq D(x, z)$

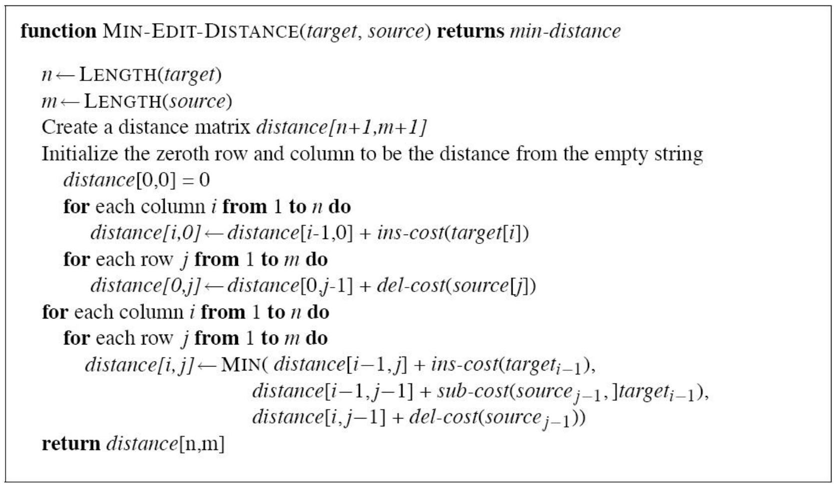
Let p_1 be the minimal edit distance path from x to y, and let p_2 be the minimal edit distance path from y to z. Then, The path $p_1 + p_2$ (not defined, but you probably get the point) is a path that transforms x to z via y, and has a cost of $D(x, y) + D(y, x)$. Therefore, there exists a path from x to z with cost $D(x, y) + D(y, x)$. The cost $D(x, z)$ of the minimal edit distance path from x to z cannot be greater than the cost of the path we have found from x to z via y:

$$D(x, z) \leq D(x, y) + D(y, x)$$

Note that the idea of "going from x to z via y" works well in the second, "transformational path-based" approach, but does not necessarily work with the two-level "alignment perspective". This second approach will not consider alignments that include the insertion *and* the deletion of a character in y, if this character does not occur in either x or z.

5 Algorithm-based proofs

To refresh our memory, here is the algorithm provided by *Jurafsky and Martin*:



```
function MIN-EDIT-DISTANCE(target, source) returns min-distance

  n ← LENGTH(target)
  m ← LENGTH(source)
  Create a distance matrix distance[n+1,m+1]
  Initialize the zeroth row and column to be the distance from the empty string
    distance[0,0] = 0
    for each column i from 1 to n do
      distance[i,0] ← distance[i-1,0] + ins-cost(target[i])
    for each row j from 1 to m do
      distance[0,j] ← distance[0,j-1] + del-cost(source[j])
  for each column i from 1 to n do
    for each row j from 1 to m do
      distance[i,j] ← MIN( distance[i−1,j] + ins-cost(target$_{i-1}$),
                           distance[i−1,j−1] + sub-cost(source$_{j-1}$,]target$_{i-1}$),
                           distance[i,j−1] + del-cost(source$_{j-1}$))
  return distance[n,m]
```

5.1 Non-negativity: $D(x, y) \geq 0$

For a given (*target*, *source*) string pair, the pseudo-code returns a value that is calculated in the following way:

- *distance*[0,0] is initialized with 0, a non-negative value.
- All other elements in the *distance* matrix are computed by summing up non-negative numbers, since
  - by the assumptions above, all costs are non-negative, and
  - the minimum of three non-negative numbers is also non-negative.
- An element of this matrix (namely, its upper right cell) is returned.

To be more precise, we show by induction that each cell of the dynamic programming table (the *distance* matrix) is non-negative. If will follow that its (n, m) cell, the value returned by the algorithm, is also non-negative.

First, the line `distance [0,0]=0` of the algorithm guarantees that *distance*$[0, 0]$ is non-negative. Then, by induction we show that *distance*$[i,\ 0]$ is non-negative for all $i \leq n$: this fact follows from the pseudo-code line filling these cells and by the assumption that $c_i(x) > 0$. Similarly, *distance*$[0,\ j]$ is non-negative for all $j \leq m$.

Finally, we show by double strong induction that *distance*$[i,\ j]$ is non-negative for all $i \leq n$ and $j \leq m$. Suppose that for some $1 \leq i \leq n$ and for some $1 \leq j \leq m$ we already know that *distance*$[i-1,\ j-1]$, *distance*$[i-1,\ j]$ and *distance*$[i,\ j-1]$ are all non-negative. Remember also the non-negativity assumptions on the three kinds of operations costs. It follows that each of the three arguments of Min in the penultimate line of the pseudo-code is non-negative. Consequently, their minimum is also non-negative, and this is the value assigned to *distance*$[i,\ j]$.

Therefore, the upper right cell of the dynamic programming table, that is *distance*$[n,\ m]$, the value returned by the algorithm, will also be non-negative.

5.2 Symmetry: $D(x, y) = D(y, x)$

Imagine *target* and *source* are reversed in the algorithm. $D(x, y)$ is the output of the original run, and $D(y, x)$ is the output of the new run. How does this new run of the algorithm relate to the original run?

First, observe that n and m will be assigned reversed values in the new run. Then, the zeroth column will now be initialized as earlier the zeroth row, and vice-versa. Subsequently, the algorithm will build up a table that is the "mirror" of the original table, by mirroring it around its diagonal: the new *distance*$[i,j]$ will be equal to *distance*$[j,i]$ of the original. In particular, the new *distance*$[n,m]$ will be equal to the original *distance*$[m,n]$.

However, as mentioned above, n and m will be assigned the reversed values, compared to the original run. Consequently, the output value of the new run, *distance*$[n,m]$, will be equal to *distance*$[n,m]$ in the original run, which is the output of the original run. Hence, $D(x, y)$, as computed by the original run, equals $D(y, x)$, as computed by the new run.

5.3 Coincidence axiom (1): if $x = y$, then $D(x, y) = 0$

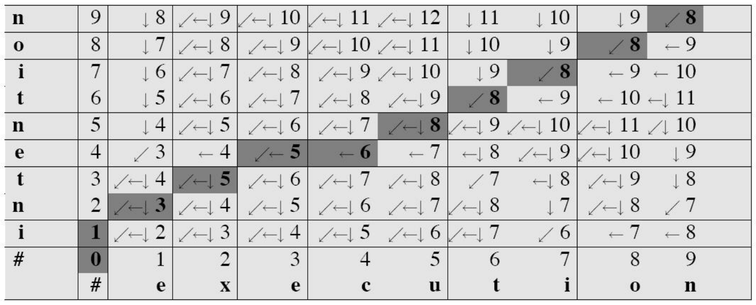
If the two arguments of the algorithm, *target* and *source* are the same, then $n = m$. Moreover, $target_i = source_i$. By the assumptions on substitution costs, $c_s(source_i, target_i) = 0$.

Now, we show by induction that running the algorithm with equal input arguments will result in *distance*$[i,i] = 0$ for all i. Indeed, this proposition is true for $i = 0$ due to the initialization of *distance*$[0,0]$ as 0. Moreover, if it also applies to *distance*$[i-1,i-1]$, then *distance*$[i-1,i-1] + c_s(source_{i-1}, target_{i-1}) = 0$. We have earlier shown that all cells in the *distance* matrix are non-negative, whereas insertion and deletion costs are positive by the assumptions. Therefore, when *distance*$[i,i]$ is calculated in the penultimate line of the code, it will be the minimum of zero and two positive numbers, that is, it will be equal to 0.

It follows that the return value, *distance*$[n, m = n]$, will also be 0.

5.4 Coincidence axiom (2): if $D(x,y) = 0$, then $x = y$

First observe that if $i \neq j$, then *distance*$[i,j]$ cannot be zero. To understand why, recall the following dynamic programming chart from your textbook:



n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8
#	**0**	1	2	3	4	5	6	7	8	9
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

The "backtrace" from *distance*$[i,j]$ to *distance*$[0,0]$, which we obtain by following the small arrows, will show the minimal edit operations needed to align the first i characters of the first string to the first j characters of the second string. If $i \neq j$, then the backtrace will contain at least one horizontal or vertical arrow, corresponding to at least one insertion or deletion, which have by assumption a positive cost. Therefore, the value of *distance*$[i,j]$, which is the sum of the costs associated with these arrows, will always be positive.

Subsequently, suppose that the algorithm has returned $D(x,y) = 0$ for some input arguments x and y. That is, *distance*$[n,m]$= 0. Therefore, n must be equal to m: the two input strings are of equal length.

The value of *distance*$[n,m]$ was calculated as the minimum of two positive values (the positive *distance*$[n-1,m]$ plus some positive insertion cost, and the positive *distance*$[n,m-1]$ plus some positive deletion cost), as well as of *distance*$[n-1,m-1]$ plus some substitution cost. This third value must be zero for their minimum to be zero. Given the assumptions on substitution costs, this third value can only be zero if *distance*$[n-1,m-1]$= 0, and also $target_{n-1} = source_{m-1}$.

Similarly, we can demonstrate by backward induction that all the elements in the diagonal *distance*$[i,i]$= 0, and $target_i = source_i$. But if all characters of the two strings are the same, then the two strings are the same: $x = y$

5.5 Triangle inequality: $D(x,y) + D(y,x) \geq D(x,z)$

The train of thought presented under the "informal" and "mathematical" approaches both rely on a combination of the "path" from x to y with the "path" from y to z. However, this combination is not always considered by the Minimal Edit Distance Algorithm, as mentioned at the end of subsection 4.4. Therefore, the idea used there cannot be adopted here.

If, nevertheless, you can demonstrate the triangle inequality by referring to the algorithm, please feel free to let me know.

6 Mathematical proofs

First, we have to demonstrate that if a and $b \in \Sigma^*$, then there exists at least one alignment from a to b. Indeed, we can construct such a (finite) alignment: let us first gradually delete the characters in a, and after arriving to the empty string, let us gradually insert the characters of b.

Subsequently, we have to demonstrate that there exists at least one alignment from a to b whose cost is $D(a, b)$.[1] This fact follows from Σ being finite, thanks to which $\{c_i(x)|x \in \Sigma\} \cup \{c_d(x)|x \in \Sigma\} \cup \{c_s(x, y)|x, y \in \Sigma\} \subset \mathbb{R}_0^+$, (the set of elementary operation costs) is also finite. A possible alignment cost is a linear combination of the elementary operation costs, with non-negative integer coefficients: the cost of inserting x n times, plus the cost of deleting x m times, plus the cost of inserting y k times, plus the cost of substituting x for y l times, etc. It follows that the set of possible alignment costs is a well-ordered subset of $\mathbb{R}_0^+$. The distance $D(x, y)$ is defined as the minimum of the set $\{C(x_0, x_1, \ldots, x_n)|(x_0, x_1, \ldots, x_n) \text{ is an alignment from } a \text{ to } b\}$, which is a subset of the set of possible alignment costs, and therefore it contains its least element. The alignment yielding this least element will be called the *minimum edit distance alignment.*

6.1 Non-negativity: $D(x, y) \geq 0$

Definitions 1 to 3 of the appendix guarantee D_i, D_d and D_s to take non-negative values. Definition 4 guarantees that $D(a, b)$ is non-negative, iff a and b are neighbors. By definition 5, the *cost* of any alignment is the sum of zero or more non-negative numbers, which is again a non-negative number. Finally, definition 6 introduces the *minimal edit distance* of two strings as the minimum of non-negative numbers, which must be again a non-negative number.

6.2 Symmetry: $D(x, y) = D(y, x)$

First, one can easily demonstrate using definitions 1 and 2 that if a is an insertion-neighbor of b, then b is a deletion-neighbor of a. Given the assumptions on insertion and deletion costs, $D_i(a, b) = D_d(b, a)$. Vice-versa, if a is a deletion-neighbor of b, then b is an insertion-neighbor of a, and $D_d(a, b) = D_i(b, a)$. Moreover, if a is a substitution-neighbor of b, then b is also a substitution-neighbor of a. Given the symmetry assumption on the substitution costs, $D_s(a, b) = D_s(b, a)$.

[1]Is this not self-evident? Suppose that in geometry we define the distance of two shapes as the (Eucledian) distance of their two closest points. Formally, let the distance of A and B be the greatest lower bound of the (Eucledian) distances $d(a, b)$ for all $a \in A$ and $b \in B$: $D(A, B) = \min\{d(a, b)|a \in A, b \in B\}$. Is it true that in such case there will always be some $a^* \in A$ and some $b^* \in B$ such that $D(A, B) = d(a^*, b^*)$? No, it is not. Imagine for instance two open half-planes, not including the straight line separating them. Two points of the two half-planes can have any small positive distance. The minimum of such distances will be zero, and so the distance of the two half-planes must be zero. And yet, you cannot finds points in the two half-planes so that their distance be precisely zero.

Consequently, by definition 4, if a is a neighbor of b, then b is a neighbor of a. Moreover, $D(a, b) = D(b, a)$.

Let $(x_0, x_1, \ldots, x_n)$ be an alignment from a to b. Then it follow from the previous paragraph that $(x_n, \ldots, x_1, x_0)$ is an alignment from b to a, and $C(x_0, x_1, \ldots, x_n) = C(x_n, \ldots, x_1, x_0)$ (by the arithmetic properties of addition). If $(x_0, x_1, \ldots, x_n)$ is *not* an alignment from a to b, then $(x_n, \ldots, x_1, x_0)$ is *not* an alignment from b to a.

Consequently, the set of alignments from y to x can be obtained by considering all the alignments from x to y and by reversing them. The set of the costs of the alignments from x to y is the same set as the set of the costs of the alignments from y to x. Consequently, $D(y, x)$ is calculated as the minimum of the same set of costs, as the set of costs that is used to calculate $D(x, y)$. Hence, $D(x, y) = D(y, x)$.

6.3 Coincidence axiom: if $x = y$, then $D(x, y) = 0$

If $x = y$, then (x) is an alignment from x to y, and its cost is zero. The cost of the minimal edit alignment cannot be greater than the cost of the alignment we have thus constructed: $D(x, y) \leq C(x) = 0$. Yet, by non-negativity, $D(x, y) \geq 0$. Therefore, $D(x, y) = 0$.

6.4 Coincidence axiom: if $D(x, y) = 0$, then $x = y$

As mentioned above, $D(x, y)$ is not only the minimum of the alignment costs, but it is also the cost of some alignment. Due to non-negativity and definition 5, the cost of this alignment can only be zero if

- either the sum in definition 5 has no addend, because the alignment consists of a single element (x_0),
- or all the addends in the summation are zero.

In the former case, by definition 5, $x = x_0$ and $y = x_0$, and hence, $a = b$. In the latter case, $D(x_i, x_{i+1}) = 0$ for all i. But observe that the positivity assumptions on the insertion and deletions costs prevent $D_i(a, b)$ and $D_d(a, b)$ to be zero. Therefore, x_i and x_{i+1} must be substitution-neighbors for all i, with zero-cost substitution. That is, by definition 3, x_i and x_{i+1} share the same prefix and the same suffix, between which x_i contains σ_1 and x_{i+1} contains σ_2. Moreover, $c_s(\sigma_1, \sigma_2) = D(x_i, x_{i+1}) = 0$. By the assumptions on the substitution costs, we obtain that $\sigma_1 = \sigma_2$. Therefore, $x_i = x_{i+1}$ for all i. In turn, $x_i = x_j$ all along the alignment. And since $x = x_0$ and $y = x_n$ in the alignment, it follows that $x = y$.

6.5 Triangle inequality: $D(x, y) + D(y, x) \geq D(x, z)$

Let $(x_0, \ldots, x_n)$ be a minimal edit distance alignment from x to y, that is, $C(x_0, \ldots, x_n) = D(x, y)$. By the definition of an alignment, $x = x_0$ and $y = x_n$.

Let also $(y_0, \ldots, y_k)$ be a minimal edit distance alignment from y to z, that is, $C(y_0, \ldots, y_k) = D(y, z)$. By the definition of an alignment, again, $y = y_0$ and $z = y_k$.

Now consider the following: $(x_0, \ldots, x_n = y = y_0, \ldots, y_k)$. It is easy to see that this is an alignment from x to z. Therefore,

$$C(x_0, \ldots, x_n = y_0, \ldots, y_k) \geq \min\{\text{costs of all alignments from } x \text{ to } z\} = D(x, z)$$

Moreover,

$$C(x_0, \ldots, x_n = y_0, \ldots, y_k) = C(x_0, \ldots, x_n) + C(y_0, \ldots, y_k) = D(x, y) + D(y, z)$$

Therefore, $D(x, y) + D(y, z) \geq D(x, z)$.

7 The assumptions are necessary, not only sufficient

As the proofs above have demonstrated it, the assumptions introduced in section 3 are sufficient for Minimal Edit Distance to be a distance metric. Some of you have also shown that most of them are also necessary conditions (provided Σ is a non-empty finite alphabet, and we wish to define Minimal Edit Distance on the entire Σ^*):

- Suppose there is some $x \in \Sigma$ such that $c_i(x) = 0$. Then, the Minimal Edit Distance of the empty string to the one-character-long string x will be at most null: inserting the character x into the empty string creates the string x, which is a zero-cost alignment between the two strings. The alignment realizing the Minimal Edit Distance cannot be more costly than the cost of this alignment. That is, $D(\epsilon, x) \leq c_i(x) = 0$. This fact contradicts the coincidence axiom (or non-negativity, if some alternative alignment is even less costly).
- Similarly, if there is some $x \in \Sigma$ such that $c_d(x) = 0$, then $D(x, \epsilon) \leq c_d(x) = 0$, contradicting the coincidence axiom (or non-negativity).
- Suppose there is some $x \in \Sigma$ such that $c_i(x) < 0$. Then $D(\epsilon, x) \leq c_i(x) < 0$, contradicting non-negativity.
- Similarly, if there is some $x \in \Sigma$ such that $c_d(x) < 0$, then $D(x, \epsilon) \leq c_d(x) < 0$, contradicting non-negativity.
- Suppose there is some x and $y \in \Sigma$ such that $c_s(x, y) < 0$. Then, let us calculate the Minimal Edit Distance of the two, one-character-long strings x and y: $D(x, y) \leq c_s(x, y) < 0$. But this would contradict non-negativity.

- Suppose there is some x and $y \in \Sigma$ such that $x \neq y$ and $c_s(x, y) = 0$. Then, again, by observing that the one-character string x can be transformed into the one-character string y by a single substitution operation, we obtain $D(x, y) \leq c_s(x, y) = 0$. But this would again contradict either the coincidence axiom, or non-negativity.

Before discussing the case $c_s(x, x) > 0$, we have to clarify whether no operation is an alternative to replacing x with x. Suppose for instance that we are computing the distance of string x from the string x. How can we align them? Solutions include: (1) no operation at all (cost is 0), (2) replace character x with character x (cost is $c_s(x, x)$), (3) delete character x and insert character x (cost is $c_d(x) + c_i(x)$), (4) insert and delete any other character(s) beside applying one of the previous options. Given our results above, we already know that the cost of (2) must be non-negative, and the costs of (3) and (4) must be positive. If we want Minimal Edit Distance to be a distance metric, then the distance of the string x from the string x must be null; hence, (3) and (4) cannot provide the optimal alignment. If we follow the informal introduction of Minimal Edit Distance, then no operation at all is an option, alignment (1) will provide the correct distance, and we can freely suppose $c_s(x, x)$ to be positive. If, however, we follow the algorithm, then each character is made to correspond to some other character, no operation is not an option, and $c_s(x, x)$ must be null for the coincidence axiom to hold.

Finally, is $c_i(x) = c_d(x)$ a necessary condition for the edit distance to be a distance metric? Technically speaking not. Imagine that $c_i(x) > c_d(x)$, but there exists a $y \in \Sigma$ such that $c_d(x) = c_i(y) + c_s(y, x)$. In this case, it is less costly to insert a y, and then to replace it with an x, than to insert x directly. Moreover, although $c_i(x) = c_d(x)$ does not hold formally, still the cost of deleting x is *de facto* equal to the costs of inserting x (via a temporary y). Note that while inserting an x via a y is possible in the "series of operations" approach (e.g., my formal definitions), it is not always possible in the "alignment" approach (e.g., using the algorithm).

8 Summary

I hope that I could show you how a seemingly simple question (whether the *Minimal Edit Distance* is a distance metric) can be approached in different ways, and what subtle problems it raises. For instance, we have seen that the alignment approach and the transformational path approach might behave surprisingly differently.

I also hope that your mathematical skills (very useful if you become either a programmer or a linguist) have been trained. You have seen how a proof by contradiction can be reformulated as a direct proof, and what minuscule details should be checked before a proof can be made work.

Any feedback on this note (as any feedback in general) will be very much appreciated, either addressed directly to me, or via Jen.

Tamás Biró