Cross-Domain Approximate String Matching

Daniel Lopresti
Gordon Wilfong

Bell Labs, Lucent Technologies
Murray Hill, NJ 07974

Approximate string matching is an important paradigm in domains ranging from speech recognition to information retrieval and molecular biology. In this paper, we introduce a new formalism for a class of applications that takes two strings as input, each specified in terms of a particular domain, and performs a comparison motivated by constraints derived from a third, possibly different domain. This issue arises, for example, when searching multimedia databases built using imperfect recognition technologies (*e.g.*, speech, optical character, and handwriting recognition). We present a polynomial time algorithm for solving the problem, and describe several variations that can also be solved efficiently.

1 Introduction

Approximate string matching is a widely-studied paradigm with important applications in domains ranging from speech recognition to information retrieval and molecular biology [SK83, GD91, FBY92, Ste94, YFM96, Gus97]. A key principle in this field is the concept of string edit distance, a measure for quantifying the similarity between two strings as well as for understanding the precise ways in which related strings may differ. In its most popular formulation, three basic operations are permitted: the deletion, insertion, and substitution of individual symbols. Each of these operations is assigned a cost, and the edit distance between two strings is then defined as the cost of the least expensive sequence of operations that transforms one string into the other.

This basic model has been both specialized and extended in numerous ways, including adding new operations (*e.g.*, transpositions, block motion), generalizing from simple strings to formal languages (*e.g.*, regular and context-free languages), and editing other types of data structures (*e.g.*, trees, 2-D strings). Another fertile area for research has been to develop techniques for performing the edit distance computation faster.

One particularly interesting class of applications involves taking strings specified in one domain and performing a comparison motivated by constraints from another. A good example of this is the case of text strings – words or sentences – that are specified, say, by their ASCII encodings, but which we would like to compare in terms of the way they are pronounced. For instance, the strings "through" and "threw" differ significantly in their ASCII representations, so a substantial amount of editing would be required to transform one into the other. Their pronunciations in English, however, are identical ($thr\overline{oo}$). Hence, we would like to be able to say that while their textual edit distance is large, their phonetic edit distance is small.

The situation becomes even more complex when we combine this with string editing for the purpose of modeling error processes. Consider, for example, the problem of querying via voice a database that was created from faxed documents. To accomplish this task, we must contend with ASR errors from the speech recognition process, a completely different class of errors from the OCR process, and the issue of judging the similarity between spoken and printed keywords. This is illustrated in Figure 1.

An analogous phenomenon can be seen in molecular biology. Table 1 presents the Genetic Code first put forth by H. Gobind Khorana in June of 1966. Proteins are generated from DNA via an

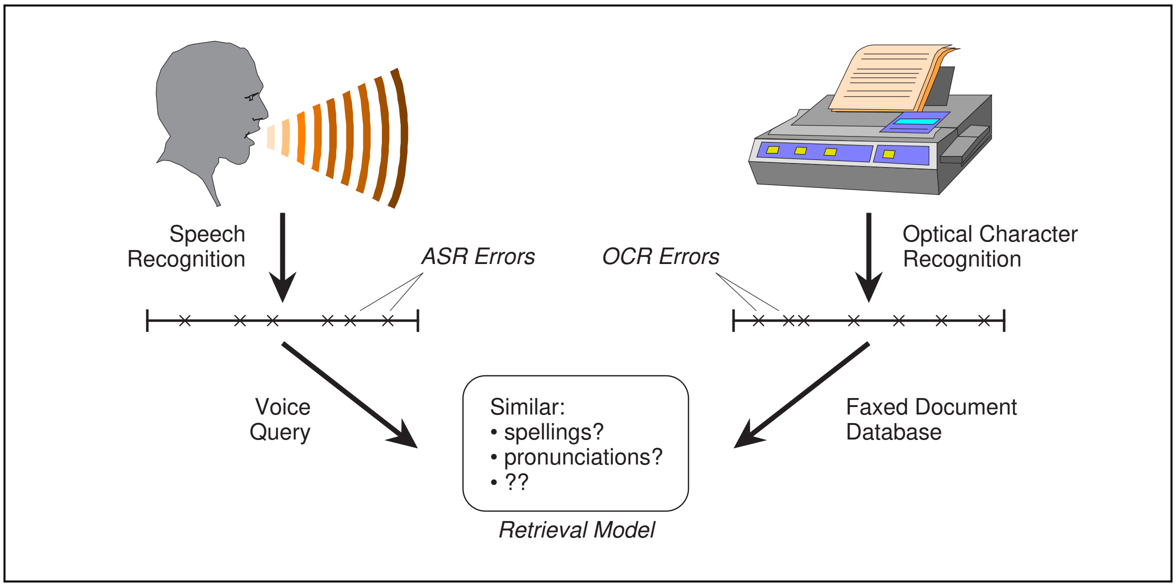

Figure 1: An example of cross-domain retrieval.

intricate chemical decryption algorithm involving messenger RNA. As indicated in the table, it takes three RNA nucleotides, or a *codon*, to specify which amino acid should be added next to the protein chain. Note that an amino acid can have several different encodings into terms of nucleotides.

UUU	Phenyl-	UCU	Serine	UAU	Tyrosine	UGU	Cysteine
UUC	alanine	UCC		UAC		UGC	
UUA	Leucine	UCA		UAA	Stop	UGA	Stop
UUG		UCG		UAG		UGG	Tryptophan
CUU		CCU	Proline	CAU	Histidine	CGU	Arginine
CUC		CCC		CAC		CGC	
CUA		CCA		CAA	Glutamine	CGA	
CUG		CCG		CAG		CGG	
AUU	Isoleucine	ACU	Threonine	AAU	Asparagine	AGU	Serine
AUC		ACC		AAC		AGC	
AUA		ACA		AAA	Lysine	AGA	Arginine
AUG	Methionine	ACG		AAG		AGG	
GUU	Valine	GCU	Alanine	GAU	Aspartic	GGU	Glycine
GUC		GCC		GAC	acid	GGC	
GUA		GCA		GAA	Glutamic	GGA	
GUG		GCG		GAG	acid	GGG	

Table 1: The Genetic Code.

Given two RNA sequences, one might naturally wonder how similar the proteins they code are. Taking this one step further as before, it is possible to imagine each of the RNA sequences first undergoing an editing process to correct for possible "noise" effects that may have occurred for any of a number of reasons, biological or otherwise (*e.g.*, mistakes in reading the sequencing gels).

Past approaches to this problem, which we call *cross-domain approximate string matching*, have been ad hoc and typically attempt to adapt the existing algorithm for string edit distance by adding special substitution costs. As we shall show in the next section, however, this may invalidate the guarantee of optimality, reducing the algorithm to the status of a heuristic.

In this paper, we introduce a formalization for cross-domain approximate string matching that captures the intentions discussed earlier. The task is phrased as a proper optimization problem, for which we present an algorithm that is guaranteed to return the best solution in polynomial time. The remainder of the paper is organized as follows. In Section 2, we present some background on traditional string editing and show why it cannot be used to solve the problem of interest. We formalize the cross-domain approximate string matching problem in Section 3, and give our algorithm for solving it in Section 4. A proof of correctness is sketched in Section 5. In Section 6, we analyze the time complexity and present a heuristic for speeding up the computation. Several variations on the problem and algorithms for their solution are described in Section 7. Finally, Section 8 offers our conclusions.

2 Approximate String Matching

We begin with some definitions. An *alphabet*, Σ, is a finite set of symbols. Define Σ^* to be the set of all finite-length sequences of symbols chosen from Σ including ε, the sequence of length 0. Each such sequence in Σ^* is a *string* over Σ. String $A_{i,j}$ is a *substring* of string $A = a_1 a_2 \ldots a_m$ consisting of the sequence of symbols $a_i a_{i+1} \ldots a_j$ for $1 \leq i \leq j \leq m$. For notational convenience, we define $A_{1,0}$ to be ε.

Say α and β are two symbols from the alphabet in question. Then we will often write the basic editing operations as $\alpha \rightarrow \varepsilon$ (the deletion of α), $\varepsilon \rightarrow \beta$ (the insertion of β), and $\alpha \rightarrow \beta$ (the substitution of β for α). The costs charged for performing these operations are $ecost(\alpha, \varepsilon)$, $ecost(\varepsilon, \beta)$, and $ecost(\alpha, \beta)$, respectively.

Let S be a set of basic editing operations that transforms string A into string B such that for any other such set S', the sum of the costs of the operations in S' is at least as large as it is in S. Then the *edit distance* between strings A and B, $edist(A, B)$, is defined to be the sum of the costs of the operations in S. Finding such a set of operations is called the *approximate string matching problem*.

This optimization problem can be solved using a well-known dynamic programming algorithm [NW70, WF74]. Let $A = a_1 a_2 \ldots a_m$ be one string and $B = b_1 b_2 \ldots b_n$ be the other. By the notation above, $edist(A_{1,i}, B_{1,j})$ is the edit distance between the first i symbols of A and the first j symbols of B. The algorithm establishes the initial conditions:

$$\begin{array}{lcl} edist(\varepsilon, \varepsilon) & = & 0 \\ edist(A_{1,i}, \varepsilon) & = & edist(A_{1,i-1}, \varepsilon) + ecost(a_i, \varepsilon) \\ edist(\varepsilon, B_{1,j}) & = & edist(\varepsilon, B_{1,j-1}) + ecost(\varepsilon, b_j) \end{array} \tag{1}$$

and the main dynamic programming recurrence is:

$$edist(A_{1,i}, B_{1,j}) = \min \left\{ \begin{array}{lcl} edist(A_{1,i-1}, B_{1,j}) & + & ecost(a_i, \varepsilon) \\ edist(A_{1,i}, B_{1,j-1}) & + & ecost(\varepsilon, b_j) \\ edist(A_{1,i-1}, B_{1,j-1}) & + & ecost(a_i, b_j) \end{array} \right. \tag{2}$$

for $1 \leq i \leq m$, $1 \leq j \leq n$. The computation builds a 2-D matrix of distance values working from the upper left corner ($edist(\varepsilon, \varepsilon)$) to the lower right ($edist(A_{1,m}, B_{1,n})$). Once it has completed, a sequence of editing decisions that achieves the optimum can be determined via backtracking. The final distance is a measure of the similarity of the two strings, and the optimal sequence of editing operations highlights the actual differences. The computation time is proportional to the product of the lengths of the two strings, $O(mn)$.

Note that for this algorithm to work, there is an underlying assumption that each index within the source string will be acted upon by at most one editing operation. Whether this assumption holds is a function of the way the edit costs are defined. If, for example, we had $ecost(\alpha, \beta) = -1$ and $ecost(\beta, \alpha) = -1$ for symbols $\alpha \neq \beta$, then the edit distance between any string containing α or β and any other string would be undefined, just as the shortest path in a graph with a negative cost cycle is undefined. (The algorithm defined by Equation 2 would still return a value, but it would not be the correct edit distance.) Assuming that all the costs are least 0 eliminates this particular problem.

One other condition is necessary, however. Consider the case of comparing the two strings "cat" and "hat" when all of the editing operations are assigned an arbitrarily large cost except for $ecost(c, f) = ecost(f, h) = 0$. Then the edit distance between "cat" and "hat" is 0, but this can only be achieved by performing two operations on the first index position of "cat": $cat \rightarrow fat \rightarrow hat$. Again, the standard dynamic programming algorithm (*i.e.*, Equation 2) does not allow for this.

What is needed is a guarantee that:

$$ecost(\alpha, \beta) \leq ecost(\alpha, \gamma) + ecost(\gamma, \beta) \tag{3}$$

for all possible combinations of symbols and editing operations. This is the well-known triangle inequality, and is an implicit assumption throughout the literature on approximate string matching. We will make use of it here as well.

Returning to the example of Section 1, consider the problem of comparing the strings "through" and "threw" in terms of their pronunciations. In the past, this problem has been attacked by incorporating multi-symbol substitutions into the edit model and specializing the edit costs to take phonetic similarities into account [Ver88, ZD96]. So, we might say $ecost(ough, ew) = 0$. Then single-symbol deletions and insertions can be used to model differences in the text domain (say, typing or OCR errors), while the multi-symbol substitutions capture phonetic relationships. As a result, the strings "through" and "threw" will be judged to have edit distance 0, while the strings "th.rough" and "threw" will have distance 1 (period insertion errors are common in OCR). The previous dynamic programming algorithm will work in this case.

There is, however, a serious conceptual problem with this approach. Consider what happens if the first string is recognized as "throu9h" by the OCR process. Mistaking a '9' for a 'g' is a common error in some systems. There is, of course, no rule for suggesting how "ou9h" should be pronounced. Hence, there is no way to know that it would be close to "ew" once the OCR error was accounted for. While we may also include a substitution operation with $ecost(9, g) = 0$, our ability to edit the substring is constrained by the fact that we are forced to make a choice: we may use either a substitution motived by OCR errors, or one motivated by pronunciations, but not both. Looking at it another way, the two low-cost substitution operations we would like to allow, $9 \rightarrow g$ and $ough \rightarrow ew$, are not compatible with the non-existent (*i.e.*, infinite-cost) operation $ou9h \rightarrow ew$; the triangle inequality is violated. Any attempt to apply the existing edit distance model and its associated dynamic programming algorithm will result in a computation no longer guaranteed to return the optimal result. Equation 2 is reduced to the status of a heuristic.

In the next two sections, we formalize this problem and present a new, optimal algorithm for its solution.

3 Cross-Domain Approximate String Matching

We think of a *domain* as a means for producing strings of symbols from a certain alphabet. Examples of domains and their alphabets include: keyboard input (the ASCII character set), printed pages of typeset text (ASCII), handwriting (ASCII), speech (60 or so phonemes in the case of English), DNA (the four nucleic acids), and proteins (the 22 amino acids). In a particular domain, the manner in which strings are produced suggests a means for judging the similarity between two strings from the domain alphabet. In the domain of scanned pages processed through optical character recognition, we might regard the two strings "baseball" and "basebal1" as being quite similar, as 'l' (el) is frequently mistaken for '1' (one) in OCR. On the other hand, in the typed-text domain "baseball" and "basebakk" might be more similar, as the 'K' key is adjacent to the 'L' on QWERTY keyboards and a typist's fingers may easily slip from one to the other.

For a given domain, we wish to capture the notion of similarity between strings as suggested by the domain by defining a model in which costs are given for editing operations that allow us to transform one string into another. More formally, we define an *edit model* E_i to consist of an alphabet Σ_i, a finite collection δ_i of basic editing operations $\alpha \rightarrow \alpha'$ each with cost $ecost_i(\alpha, \alpha')$, where $\alpha, \alpha' \in \Sigma_i^*$, and an edit distance $edist_i(A, A')$ between any two strings as defined in Section 2. The basic operations will be assumed to be single-symbol deletions, insertions, and substitutions. We will write an edit model E_i as a triple $(\Sigma_i, \delta_i, edist_i)$ defining the alphabet, the set of basic operations and their costs, and the edit distance resulting from the given basic operations and their costs.

In addition to the edit model defined by a domain, there are also transcriptions that map between domains. A *transcription model* $T_{i,j}$ consists of two alphabets, the *source alphabet* Σ_i and the *target alphabet* Σ_j, and a finite collection $\tau_{i,j}$ of basic transcription operations (with costs) each of which can be regarded as a substitution. We denote such an operation by $\alpha \rightarrow \alpha'$ where α, called the *left hand side*, is a string in Σ_i^* and α', called the *right hand side*, is ε or a single symbol in Σ_j, and we write $tcost_{i,j}(\alpha, \alpha')$ to mean the cost of performing that substitution. A *transcription* of string $A \in \Sigma_i^*$ into a string $A' \in \Sigma_j^*$ is the action of performing a series of substitutions (*i.e.* basic transcription operations), replacing substrings of string A with 0 or 1 symbols from alphabet Σ_j to yield string A'. The cost of the transcription is just the sum of the costs of the substitutions.

Notice that there may not be any transcription from string A to string A'. For instance, if A' contains a symbol α, but α is not a symbol in any transcription's right hand side, then there is no transcription from any string A to A'. Therefore the *transcription distance* between two strings A and A', $tdist_{i,j}(A, A')$, is defined to be ∞ if there is no transcription from A to A' or the minimum cost of any transcription of A into A' in the case that one or more transcriptions from A to A' exist. Each transcription model $T_{i,j}$ will be written as a 4-tuple $(\Sigma_i, \Sigma_j, \tau_{i,j}, tdist_{i,j})$ where Σ_i is the source alphabet, Σ_j is the target alphabet, $\tau_{i,j}$ is the set of allowable substitutions (and their costs), and $tdist_{i,j}$ is the transcription distance defined by the costs of the substitutions.

Note that the definitions of edit and transcription models are quite similar in that they involve transforming one string into another at some cost. The key difference is that edit models transform strings from the same alphabet, whereas transcription models, in general, transform strings from one alphabet to another.

The *cross-domain approximate string matching problem* is, given:

1. domains $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ with associated edit models $E_1 = (\Sigma_1, \delta_1, edist_1)$, $E_2 = (\Sigma_2, \delta_2, edist_2)$, and $E_3 = (\Sigma_3, \delta_3, edist_3)$,
2. two strings, $A \in \Sigma_1^*$ from $\mathcal{D}_1$ and $B \in \Sigma_2^*$ from $\mathcal{D}_2$,
3. transcription models $T_{1,3} = (\Sigma_1, \Sigma_3, \tau_{1,3}, tdist_{1,3})$ and $T_{2,3} = (\Sigma_2, \Sigma_3, \tau_{2,3}, tdist_{2,3})$,

determine:

$$xdist_{1,2,3}(A, B) = \min_{\substack{A' \in \Sigma_1^*,\ B' \in \Sigma_2^*, \\ A'',\ B'' \in \Sigma_3^*}} \left[\begin{array}{c} edist_1(A, A') \ + \ tdist_{1,3}(A', A'') \ + \\ edist_2(B, B') \ + \ tdist_{2,3}(B', B'') \ + \\ edist_3(A'', B'') \end{array} \right] \tag{4}$$

Equation 4 defines the optimal way to match string A to string B by first editing each in its own domain, then transcribing both resulting strings into the third domain, and finally comparing the transcriptions using an edit model appropriate for that domain. As such, it captures the notion of cross-domain string matching discussed in Section 1. The computation is perhaps more easily visualized by considering Figure 2.

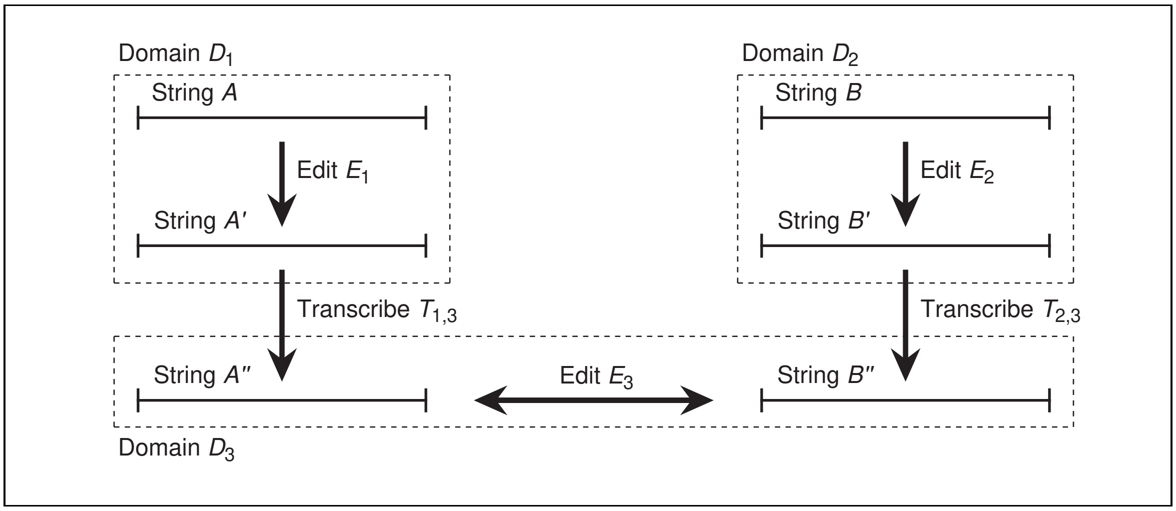

Figure 2: The cross-domain approximate string matching problem.

4 Algorithms

A dynamic programming scheme will be defined to solve the minimization problem of Equation 4. As a reminder, we have three domains $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ involved in this problem, defining edit models $E_1 = (\Sigma_1, \delta_1, edist_1)$, $E_2 = (\Sigma_2, \delta_2, edist_2)$ and $E_3 = (\Sigma_3, \delta_3, edist_3)$ respectively. Also there are two transcription models $T_{1,3} = (\Sigma_1, \Sigma_3, \tau_{1,3}, tdist_{1,3})$ and $T_{2,3} = (\Sigma_2, \Sigma_3, \tau_{2,3}, tdist_{2,3})$. As mentioned, each transcription $A' \rightarrow A''$ in $\tau_{1,3}$ and each transcription $B' \rightarrow B''$ in $\tau_{2,3}$ is such that the length of A'' and B'' is 0 or 1. In addition, we will assume that $\varepsilon \rightarrow \varepsilon$ is in both $\tau_{1,3}$ and $\tau_{2,3}$ and that this operation has cost 0.

Let $A = a_1, a_2, \ldots, a_m$ be a string in Σ_1^* and $B = b_1, b_2, \ldots, b_n$ a string in Σ_2^*. We wish to determine $xdist_{1,2,3}(A, B)$.

Consider first the case that A and B each correspond to a single left hand side in some transcription. Thus the distance between them should include the edit cost of transforming A into a string A' that is a left hand side of some transcription $A' \rightarrow A''$ in $\tau_{1,3}$, the edit cost of transforming B into a string B' that is the left hand side of some transcription $B' \rightarrow B''$ in $\tau_{2,3}$, the cost of transcribing A' to A'' in Σ_3^*, the cost of transcribing B' to B'' in Σ_3^*, and the edit distance between A'' and B''. This is depicted in Figure 3. Thus we define:

$$xcost_{1,2,3}(A, B) = \min_{\substack{A' \rightarrow A'' \in \tau_{1,3}, \\ B' \rightarrow B'' \in \tau_{2,3}}} \left[\begin{array}{c} edist_1(A, A') \; + \; tcost_{1,3}(A', A'') \; + \\ edist_2(B, B') \; + \; tcost_{2,3}(B', B'') \; + \\ ecost_3(A'', B'') \end{array} \right] \tag{5}$$

Note the similarity between the formulations of Equations 5 and 4.

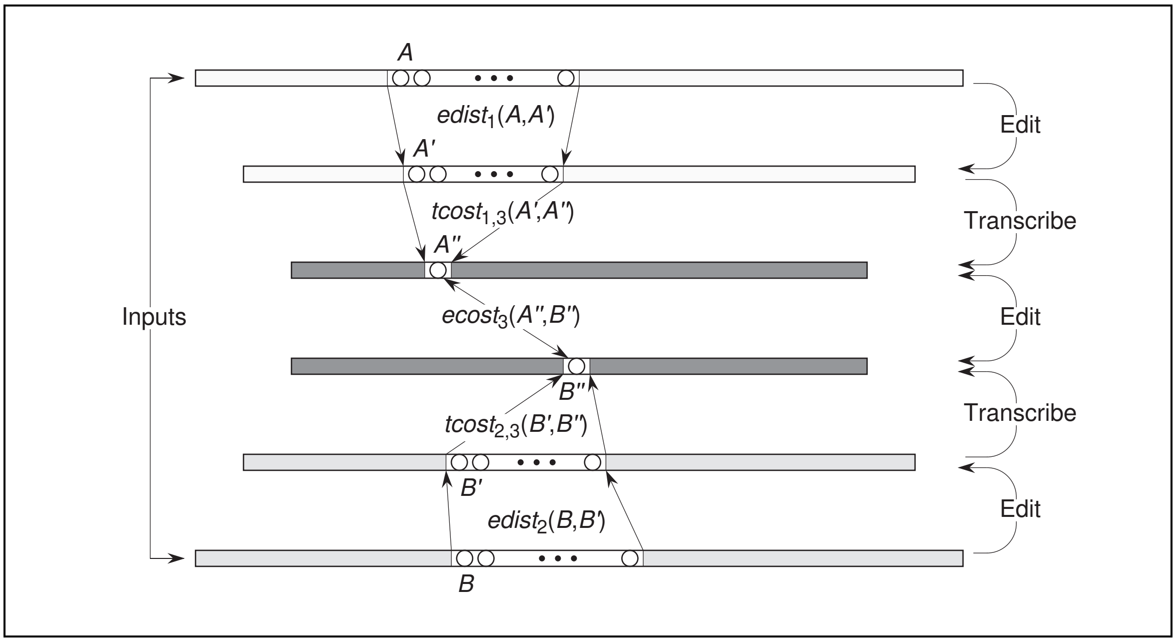

Figure 3: Depiction of the *xcost* computation.

Recall that $A_{1,i}$ refers to the substring consisting of the first i symbols of A, and $B_{1,j}$ to the substring consisting of the first j symbols of B. Now we can define $xdist_{1,2,3}(A,B)$, the cross-domain distance between A and B, by defining $xdist_{1,2,3}(A_{1,i},B_{1,j})$ recursively as:

$$xdist_{1,2,3}(A_{1,i},B_{1,j}) = \min \begin{cases} \min\limits_{1\le k\le i} \left[\begin{array}{r} xdist_{1,2,3}(A_{1,i-k},B_{1,j}) + \\ xcost_{1,2,3}(A_{i-k+1,i},\varepsilon) \end{array} \right], \\ \min\limits_{1\le k'\le j} \left[\begin{array}{r} xdist_{1,2,3}(A_{1,i},B_{1,j-k'}) + \\ xcost_{1,2,3}(\varepsilon,B_{j-k'+1,j})] \end{array} \right], \\ \min\limits_{1\le k\le i,\, 1\le k'\le j} \left[\begin{array}{r} xdist_{1,2,3}(A_{1,i-k},B_{1,j-k'}) + \\ xcost_{1,2,3}(A_{i-k+1,i},B_{j-k'+1,j}) \end{array} \right] \end{cases} \tag{6}$$

where we define the base case to be:

$$xdist_{1,2,3}(\varepsilon,\varepsilon) = xcost_{1,2,3}(\varepsilon,\varepsilon). \tag{7}$$

That is, we test the hypothesis that the last left hand side of a substitution in $A_{1,i}$ is of length k and that in $B_{1,j}$ is of length k'. The cost of this hypothesis is the sum of the costs of matching these two left hand sides and the earlier-computed cost of doing the cross-domain matching of the substrings $A_{1,i-k}$ and $B_{1,j-k'}$. Then *xdist* is the minimum of all such hypotheses. Note the similarity between the formulations of Equations 6 and 2.

5 Proof of Correctness

A solution to Equation 4 defines two strings A'' and B'' of Σ_3^*. We can represent A'' as $\alpha = a_1a_2\dots a_s$ and B'' as $\beta = b_1b_2\dots b_t$ where each a_i is either a symbol of A'' or ε, each b_i is either a symbol of B'' or ε, and the symbols that are not ε appear in the same order as they do in A'' and B''. That is, α is just A'' with ε between some of the symbols of A'' and similarly for β and B''. We now discuss how we choose where we place each ε.

Note that a solution to $edit_3(A'', B'')$ determines a set of operations on A'' that transforms it into B'' with total cost $edit_3(A'', B'')$. These operations in turn define α and β with the property that a_i and b_i are both symbols of Σ_3 if b_i was substituted for a_i, a_i is ε and $b_i \in \Sigma_3$ if b_i was inserted, and finally $a_i \in \Sigma_3$ and $b_i = \varepsilon$ if a_i was deleted. The representations α and β together are called an *alignment* [SK83].

In an optimal solution to Equation 4, the transformation from A' to A'' consists of transcriptions of the form $A'_{i,j} \rightarrow a_k$ or $A'_{i,j} \rightarrow \varepsilon$. Note, however, that there might be some $a_k = \varepsilon$ with no corresponding transcription $A'_{i,j} \rightarrow \varepsilon$ in an optimal solution. That is, a_k is in the alignment only due to the computation of $xdist_3(A'', B'')$ and not due to any transcription. However, we have assumed that there is a transcription of the form $\varepsilon \rightarrow \varepsilon$ in $\tau_{1,3}$ and that the cost of the transcription is 0. Thus, there is another optimal solution to Equation 4 in which the 0 cost transcription $\varepsilon \rightarrow \varepsilon$ is added to "produce" a_k in α. Therefore there is an optimal solution S for Equation 4 with the property that for every a_k in α there is some transcription $A'_{i,j} \rightarrow a_k$ in the solution. Similarly, we can assume that there is a transcription in S of the form $B_{r,s} \rightarrow b_k$ for every b_k in β.

In this optimal solution S, for each left hand side $A'_{i,j}$ or $B'_{r,s}$ of a transcription used there is some corresponding $A_{a,b}$ such that $A_{a,b}$ is edited into $A'_{i,j}$ or some corresponding $B_{c,d}$ that is edited into $B'_{r,s}$. Examining Equation 6 shows that all such partitionings of A and B into segments that are in turn edited into left hand sides of transcriptions are considered in searching for a minimum cost solution and so, in particular, S is considered by Equation 6 in searching for a minimum cost solution.

Theorem 1 *The computation of $xdist_{1,2,3}(A, B)$ as given by Equation 6 produces a solution to the cross-domain approximate string matching problem defined in Equation 4.*

6 Time Complexity

Since we have assumed that the edit models consist of single-symbol insertions, deletions, and substitutions, the standard dynamic programming procedure described in Section 2 is applicable. Define the $(m+1) \times (n+1)$ matrix M where $M(i, j)$ is the value of $xdist_{1,2,3}(A_{1,i}, B_{1,j})$. Then computing $xdist_{1,2,3}(A_{1,m}, B_{1,n})$ is a matter of iteratively filling the entries of M in order from smaller indices to larger ones, since the value of $xdist_{1,2,3}(A_{1,i}, B_{1,j})$ depends only the values of $M(i', j')$ where $0 \leq i' < i$ and $0 \leq j' < j$ as seen in Equation 6.

Let t_i denote the number of transcriptions in $\tau_{i,3}$ for $i = 1, 2$. Define $\alpha_{i,j}$ to be the left hand side of a transcription in $\tau_{i,3}$ for $i = 1, 2$ and $1 \leq j \leq t_i$. Suppose the length of the longest left hand side of any transcription in $\tau_{i,3}$ is s_i, $i = 1, 2$. The value of $edist_1(A_{i,m}, \alpha_{1,j})$ can be computed in time $O(ms_1)$ using the standard dynamic programming method mentioned in Section 2. The computation of $edist_1(A_{i,m}, \alpha_{1,j})$ actually produces the values of all $edist_1(A_{i,k}, \alpha_{1,j})$ for $i \leq k \leq m$. Thus, producing all the values $edist_1(A_{i,k}, \alpha_{1,j})$ for $1 \leq i \leq k \leq m$ and $1 \leq j \leq t_1$ can be done in $O(m^2 s_1 t_1)$ time. Similarly, the value of each $edist_2(B_{i,k}, \alpha_{2,j})$ for $1 \leq i \leq k \leq n$ and $1 \leq j \leq t_2$ can be computed in $O(n^2 s_2 t_2)$ time. The total time $\mathcal{T}_1$ for this preprocessing step is then $O(m^2 s_1 t_1 + n^2 s_2 t_2)$.

Using these precomputed values, we can compute any $xcost_{1,2,3}(A_{i,j}, B_{k,l})$ in time $O(t_1 t_2)$ and so we can compute all such values for $1 \leq i \leq j \leq m$ and $1 \leq k \leq l \leq n$ in time $\mathcal{T}_2 = O(m^2 n^2 t_1 t_2)$. Now using these precomputed values, we can compute each value $xdist_{1,2,3}(A_{1,i}, B_{1,j})$ in $O(mn)$ time, and so computing all such values can be done in time $\mathcal{T}_3 = O(m^2 n^2)$. To summarize, using

$$O(m^2 s_1 t_1 + n^2 s_2 t_2 + m^2 n^2 t_1 t_2)$$

preprocessing time we can compute $xdist_{1,2,3}(A_{1,m}, B_{1,n})$ in $O(m^2n^2)$ additional time. For a specific set of edit and transcription models, the values s_1, s_2, t_1 and t_2 are considered fixed constants and so the running time for computing $xdist_{1,2,3}(A_{1,m}, B_{1,n})$ becomes $O(m^2n^2)$ or $O(n^4)$ when $m = n$.

One obvious heuristic for speeding up the computation would be to limit the length of the strings that we try to match to a left hand side of a transcription. In the computation shown in Equation 6 it may be sensible to limit the size of k and k' by some simple function of s_1 and s_2 respectively. For instance, it might be reasonable to restrict k to the range $[1, 2s_1]$ and k' to the range $[1, 2s_2]$. Then the time $\mathcal{T}_1$ would become $O(ms_1^2t_1 + ns_2^2t_2)$ and the time $\mathcal{T}_2$ would be $O(s_1s_2mnt_1t_2)$ since only those $xcost(A_{i,j}, B_{k,l})$ need be computed where $j - i$ is bounded by a function of s_1 and $l - k$ is bounded by a function of s_2. Finally, the time $\mathcal{T}_3$ becomes $O(s_1s_2mn)$. Thus, for fixed edit and transcription models, the time to compute $M(m, n)$ using this heuristic would be given by $O(mn)$ or $O(n^2)$ in the case $m = n$.

7 Variations

In this section we consider two interesting variations on cross-domain string matching. These arise from restricting one or more of the degrees of freedom in the original problem (recall Figure 2). What happens, for example, if we do not allow any editing in either of the first two domains, $\mathcal{D}_1$ and $\mathcal{D}_2$? In the third domain, $\mathcal{D}_3$? These variations are depicted in Figure 4.

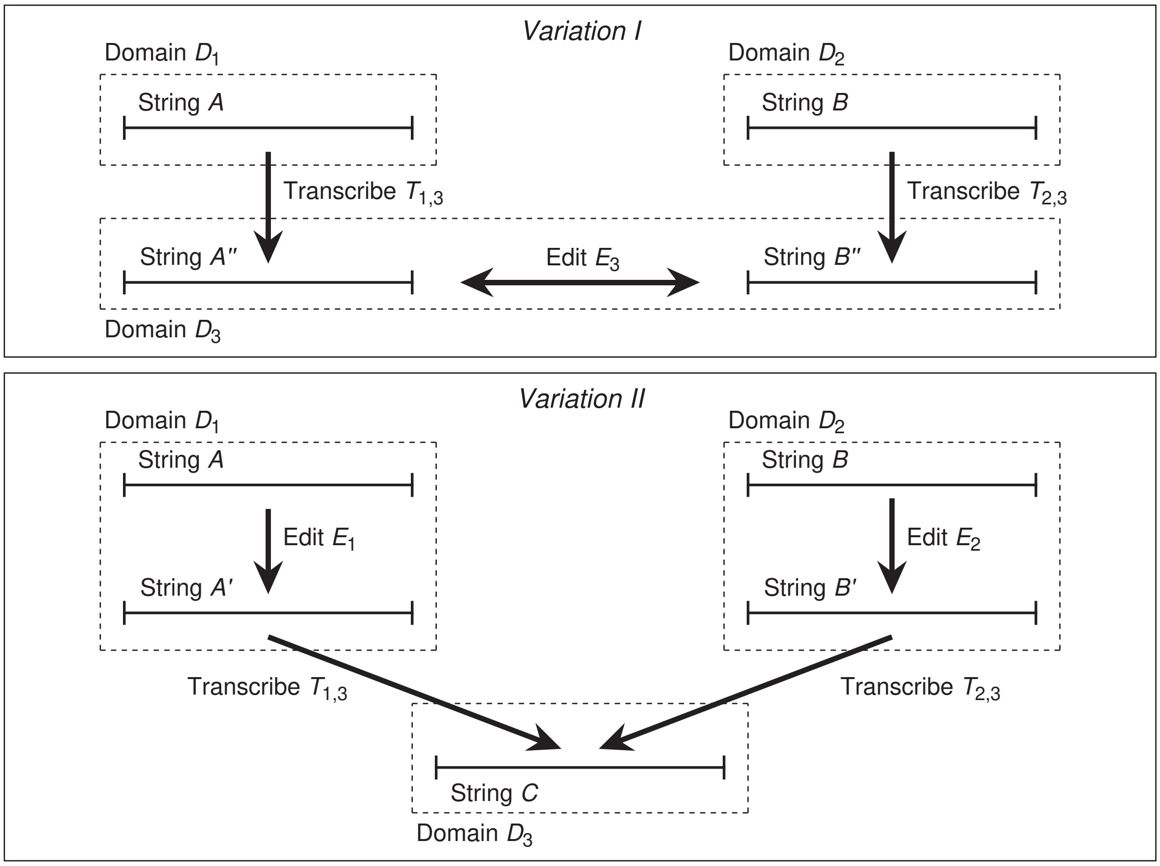

Figure 4: Variations on the cross-domain approximate string matching problem.

In the first of these cases, the problem specified in Equation 4 becomes:

$$xdist_{1,2,3}(A,B) = \min_{A'',\, B'' \in \Sigma_3^*} \left[\begin{array}{r} tdist_{1,3}(A,A'') + \\ tdist_{2,3}(B,B'') + \\ edist_3(A'',B'') \end{array} \right] \tag{8}$$

This is solved by changing $xcost$ to be:

$$xcost_{1,2,3}(A,B) = \min_{\substack{A \rightarrow A'' \in \tau_{1,3}, \\ B \rightarrow B'' \in \tau_{2,3}}} \left[\begin{array}{r} tcost_{1,3}(A,A'') + \\ tcost_{2,3}(B,B'') + \\ ecost_3(A'',B'') \end{array} \right] \tag{9}$$

with the main recurrence for $xdist$, Equation 6, defined as before. Note that, as with the heuristic presented in the previous section, the length of the strings matched to the left hand side of a transcription is inherently bounded. Following that earlier analysis, the time complexity in this case is seen to be $O(ms_1^2t_1 + ns_2^2t_2 + s_1s_2mnt_1t_2 + s_1s_2mn)$, or $O(n^2)$ when $m = n$ and t_i and s_i are constants.

This particular variation exhibits an intriguing relation to the "generalized" string matching problem [SK83] (also referred to as the "consensus sequence" problem [LZ97]). Indeed, if the edit costs in the third domain are all set to be ∞ except for exact matches which are given cost 0, and if the transcriptions in the first two domains are defined to mimic a specific set of editing operations (deletions, insertions, and substitutions), then Equations 6 and 9 solve exactly this same problem (in exactly the same time, $O(n^2)$). Hence, this particular variation of cross-domain string matching is actually a more general form of "generalized" string matching.

On the other hand, if editing is not allowed in the third domain the problem specification becomes:

$$xdist_{1,2,3}(A,B) = \min_{\substack{A' \in \Sigma_1^*,\, B' \in \Sigma_2^*, \\ C \in \Sigma_3^*}} \left[\begin{array}{r} edist_1(A,A') \;+\; tdist_{1,3}(A',C) + \\ edist_2(B,B') \;+\; tdist_{2,3}(B',C) \end{array} \right] \tag{10}$$

with:

$$xcost_{1,2,3}(A,B) = \min_{\substack{A' \rightarrow C \in \tau_{1,3}, \\ B' \rightarrow C \in \tau_{2,3}}} \left[\begin{array}{r} edist_1(A,A') \;+\; tcost_{1,3}(A',C) + \\ edist_2(B,B') \;+\; tcost_{2,3}(B',C) \end{array} \right] \tag{11}$$

and Equation 6 for $xdist$ defined as before. The time complexity in this case remains the same as in the original, unrestricted problem (*i.e.*, $O(n^4)$).

8 Conclusions

We have defined a model for the problem of comparing two strings that have been produced in (possibly) different domains, both of which have transcriptions into some common third domain. The distance measure in this model accounts for the cost to edit away any potential errors in the strings, the cost to transcribe the corrected strings into the common domain, and the edit distance between the two resulting transcribed strings. This allows us to capture accurately the edit costs in all three domains, an important feature since the edit models may vary considerably even if their underlying alphabets are the same. The method we derived for computing this distance measure between strings has been shown to have running time that is polynomially bounded.

There are a number of application areas for this type of string comparison model, as outlined in Section 1. Our future plans include implementing our algorithm and studying its effectiveness in multimedia information retrieval.

References

[FBY92] William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval: Data Structures & Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[GD91] M. Gribskov and J. Devereux. *Sequence Analysis Primer.* Stockton Press, New York, NY, 1991.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, Cambridge, UK, 1997.

[LZ97] Daniel Lopresti and Jiangying Zhou. Using consensus sequence voting to correct OCR errors. *Computer Vision and Image Understanding*, 67(1):39–47, July 1997.

[NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino-acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[SK83] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, 1983.

[Ste94] Graham A. Stephen. *String Searching Algorithms.* World Scientific, Singapore, 1994.

[Ver88] Jean Veronis. Computerized correction of phonographic errors. *Computers and the Humanities*, 22:43–56, 1988.

[WF74] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21:168–173, 1974.

[YFM96] Tieng K. Yap, Ophir Frieder, and Robert L. Martino. *High Performance Computational Methods for Biological Sequence Analysis.* Kluwer Academic Publishers, Boston, MA, 1996.

[ZD96] Justin Zobel and Philip Dart. Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 166–172, Zurich, Switzerland, August 1996.