

String Edit Analysis for Merging Databases

J. Joanne Zhu and Lyle H. Ungar
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
(215) 898-7449
jzhu@seas.upenn.edu, ungar@cis.upenn.edu

ABSTRACT

The first step prior to data mining is often to merge databases from different sources. Entries in these databases or descriptions retrieved using information extraction, may use significantly different vocabularies, so one often needs to determine whether similar descriptions refer to the same item or to different items (e.g., people or goods). String edit distance is an elegant way of defining the degree of similarity between entries and can be efficiently computed using dynamic programming (Ristad and Yianilos, 1977). However, in order to achieve reasonable accuracy, most real problems require the use of extended sets of edit rules with associated costs that are tuned specifically to each data set. We present a flexible approach to string edit distance, which can be automatically tuned to different data sets and can use synonym dictionaries. Dynamic programming is used to calculate the edit distance between a pair of strings based on a set of string edit rules including a new edit rule that allows words and phrases to be deleted or substituted. A genetic algorithm is used to learn costs corresponding to each edit rule based on a small set of labeled training data. Deleting contentless words like "method" and substituting synonyms such as "ibuprofen" for "Motrin" significantly increases the algorithm's accuracy (from 80% to 90% on a difficult sample medical data set), when costs are correctly tuned. This string edit-based matching tool is easily adapted for a variety of different cases when one needs to recognize which text strings from different information sources refer to the same item such as a person, address, medical procedure or product .

1. INTRODUCTION

When different databases with the same type of information are merged, it is often necessary to determine which entries refer to the same item and which do not. Items such as people, addresses, goods for sale and laboratory procedures in a hospital are described by different text strings in different databases. Recognizing the identity of items based on different descriptions is also important when doing data mining on the results of information extraction. Few descriptions of products extracted from the web contain unique identifiers such as ISBN numbers. Databases and other information sources are often combined, and generally contain tens of thousands of different items or more, so it is highly desirable to have an automated method of determining which entries are similar and can be combined. We address this problem by using string edit distance to determine the degree of similarity between the text strings describing the database entries. String edit distance is the total cost of transforming one string into another using a set of edit rules, each of which have an associated cost. We show how these costs can be learned for each problem domain using a small set of labeled examples of strings which refer to the same items.

One version of this problem is referred to as the merge-purge, field matching, or record linkage problem [1-6] and occurs often. For example, different mailing lists may represent one person's address in a number of ways or there may be many entries for different people in one household. When mailing lists are sold or exchanged between different companies, they need to be merged, and the process is often a tedious task. Bibliographic references are often represented differently, and need to be compared to determine if they refer to the same item [2,4]. In DNA sequencing, a DNA sequence can be represented by a string. A sequence in one DNA strand often needs to be matched to the closest sequence in a different strand. A final example is Web product descriptions. On the World Wide Web, there are many different ways to the same product. In order to compare prices for example, there must be a way to accurately determine which descriptions are for the same product.

This paper attempts to provide an effective method of determining if strings are similar by providing an algorithm to find the optimal costs of a set of string edit rules based on the characteristics of the databases to be merged. The string edit distance is then used to determine if a pair of strings are sufficiently similar to merge the items they describe. The

approach uses dynamic programming to compute edit distance and a genetic algorithm to learn their costs.

We add one key feature to string edit distance calculations: the use of extensive dictionaries of synonyms and of words which should simply be deleted ("context-specific stop words"). Consider, for example, the problem of merging mailing lists. For example, "Apartment 390" and "APT 390" are the same location. Word substitution is especially useful here if "Apartment" and "APT" are added to the substitution table. Another example is if two people live at the same place but only one mailing is required for the household. Again string edit distance can be used to determine the similarity of the addresses.

We use data from two hospital databases to motivate and to test the algorithm. In order to effectively analyze data across hospitals for data mining and analysis, medical databases that contain similar information require merging. Databases from different hospital do not contain identical descriptions for items such as procedures, drugs, or treatments. (Diagnoses are mostly standardized to ICD9 codes.) In fact, in the data we examined, each hospital database contains approximately 10,000 terms, and across ten hospitals, approximately 90,000 terms are different. Currently, a preliminary match of different item names is made using crude similarity measures, which include using the UMLS as a synonym dictionary. Proposed matches are then examined (at considerable expense) by physicians to determine true matches. We use such hand-labeled data as training and testing sets. A potentially more accurate way of doing matches is to compare text strings of the entries. This, however, requires careful tuning of the string edit distances. (E.g., "Hep A" is more similar to "Hepatitis A" than to "Hep B", which would not be true for the obvious spelling correction edit rules.) It also requires incorporation of synonym dictionaries of medical terms, and low cost of removal for domain-specific "stop words" such as "method" or "assay". This paper provides a method for extending string edit distance to incorporate these features.

2. PROCEDURES AND METHODS USED

We first address the problem of calculating the string edit distance, and then turn to the issues of tuning the costs of the edit operations. Many different edit rules can be used. Table 1 lists the set that we use in the results section below. String edit distance between a pair of strings is defined as the minimum total cost of transforming one string into the other one. We follow Ristad and Yianilos [7] in using dynamic programming to find the permutation of edit operations with the minimum total edit cost for a pair of strings.

Table 1: String Edit Rules and Costs Based on Sample Training Data

Rule	Cost ¹
Deletion:	
1.Delete a vowel	0.73

¹ This is the set of costs produced by the learning program on the sample dataset described below.

2.Delete a consonant	0.62
3.Delete a number	1.13
4.Delete white space	0.62
5.Delete a '.' (period)	1.20
6.Delete a ',' (decimal point)	0.77
7.Repeated deletion of characters	0.05

Substitution:

8.Substitute characters of the same type	1.53
9.Substitute characters of different types	1.08
10.Substitute numbers	1.61
11.Substitute or delete a word or phrase	0.73

Two strings are considered to be similar if the edit distance between a pair of strings falls below some user defined threshold. The threshold affects the edit costs learned, which in turn affects the edit distances calculated. An optimal threshold can be experimentally determined for given data sets. In the results given below, a threshold of 10% of the length of the longer of the two strings being compared was used. For example, if the two strings in question are "foo" and "foobar", the longer string, which is 6 characters in length, yields a threshold of 0.6. If the edit distance between "foo" and "foobar" is below 0.6, then these strings are considered to be similar.

3. EDIT DISTANCE CALCULATIONS

Edit distances are calculated by using dynamic programming to produce a table of string edit costs. This table implicitly contains the edit cost of every permutation of edit rules required to transform one string into another. The table is a two dimensional array whose values are calculated from the upper left diagonally to the bottom right. The upper left corner is always initialized to zero since there is no cost to transform the null string into the null string. The edit distance at each point is calculated from the surrounding values to the left and above the current position. These surrounding values represent the minimum edit distance required to reach that particular position within the two strings. The edit cost to obtain the current position from one of the surrounding positions is first calculated and then added to the value at that surrounding position. This is done because an edit rule is required to move from the previous position to the current one. Performing these operations produces three edit distances. The minimum edit distance is assigned to the current position. This algorithm guarantees that the minimum edit distance will always be the value at the lower right corner.

Edit operations are traditionally deletion, substitution or, sometimes, transposition of characters. We extend these below to include similar operations at the word level.

The algorithm is best illustrated by an example. Consider two strings from a sample database. They are "ASO TITER" and "ASO TEST" or s_1 and s_2 respectively. If we have a set of two rules, Table 2 results.

Table 2: An example of building the edit distance table with two simple edit rules.

J	→	1	2	3	4	5	6	7	8
I		A	S	O		T	E	S	T
↓		0	1	2	3	4	5	6	7
1	A	1	0	1	2	3	4	5	6
2	S	2	1	0	1	2	3	4	5
3	O	3	2	1	0	1	2	3	4
4		4	3	2	1	0	1	2	3
5	T	5	4	3	2	1	0	1	2
6	I	6	5	4	3	2	1	1	2
7	T	7	6	5	4	3	2	2	2
8	E	8	7	6	5	4	3	2	3
9	R	9	8	7	6	5	4	3	4

Rules:

Delete a character a cost of 1

Substitute a character for another character at a cost of 1

The row index is i and the column index is j and will be referenced in the form (i, j) . The upper left corner is set to zero. At $(1, 1)$ no operation is required since no rule is needed to transform 'A' into 'A'. At $(2, 1)$ there are two choices. One way is to delete the 'S' in s_1 at a cost of 1. This means that "AS" in s_1 is matched to 'A' and null in s_2 at the cost of one deletion. Another approach is to delete the 'A' in s_1 and substitute 'S' in s_1 for the 'A' in s_2 . This means that "AS" in s_1 is matched to null and 'A' in s_2 at the cost of one deletion and one substitution. This is more expensive than the first method, so the value at $(2, 1)$ is assigned to be its upper neighbor 0, which represents matching 'A' in s_1 to 'A' in s_2 , plus the cost to delete 'S' in s_1 . The row and column indices 2 and 1 can be interpreted to mean that two characters of s_1 and one character of s_2 have been processed and that 1, the value at $(2, 1)$, is the minimum edit distance for the two substrings.

The first five characters of both strings match completely. At $(5, 5)$ the edit cost is still 0. Then, the example becomes more interesting. Consider the series of operations needed to obtain a value of 2 at $(8, 6)$. At this location, "ASO TITE" in s_1 has been matched to "ASO TE" in s_2 . There are many permutations of operations that can be used to process the two strings up to this point, but not all of them produce the minimum edit distance of 2. The table effectively deals with this issue. At location $(8, 6)$, it has already considered all possibilities and has recorded that there is some permutation that produces an edit distance of 2. Backtracking in the table shows that the minimum edit distance is produced by matching the first five characters of both strings, deleting "IT" in s_1 , and matching 'E' in s_1 to 'E' in s_2 . The final result is, using the given set of two rules with the fixed cost of 1 each, the minimum edit distance for s_1 and s_2 is 4. This is the algorithm in its simplest form.

One way to significantly decrease the string edit distance is to add more rules. For this study, eleven edit rules were chosen to fit the characteristics of the test databases. One very powerful rule provides the ability to substitute entire words or phrases. A special case of this rule is the ability to delete words or phrases. This is simply the substitution of a word by the null string.

When a rule of this type is added to the previous example, Table 3 results.

Table 3: An example of building the edit distance table with two simple edit rules and a word deletion rule.

J	→	1	2	3	4	5	6	7	8
I		A	S	O		T	E	S	T
↓		0	1	2	3	4	5	6	7
1	A	1	0	1	2	3	4	5	6
2	S	2	1	0	1	2	3	4	5
3	O	3	2	1	0	1	2	3	4
4		4	3	2	1	0	1	2	3
5	T	5	4	3	2	1	0	1	2
6	I	6	5	4	3	2	1	1	2
7	T	7	6	5	4	3	2	2	2
8	E	8	7	6	5	4	3	2	3
9	R	9	8	7	6	5	4	3	4

Rules:

Delete a character at a cost of 1

Substitute a character for another character at a cost of 1

Delete the word "TEST" at a cost of 0

For this particular example, it is not immediately apparent that adding the new rule of deleting the word "TEST" did anything at all, since the minimum edit distance of the two strings is still 4. However, there are differences in the lower right side of the table. The edit distance at $(6, 8)$ is 3 in Table 2 and 2 in Table 3. At this index, "ASO TI" in s_1 and "ASO TEST" in s_2 have been processed. Again, the first five characters of both strings can be matched at no cost. Then, 'T' in s_1 can be substituted for 'E' in s_2 and "ST" in s_2 can be deleted. This gives an edit cost of 3 as shown in Table 1. If, however, "TEST" can be deleted at no cost, then only "TI" in s_1 needs to be deleted which results in an edit cost of 2 shown in Table 3.

In another example where s_1 and s_2 are "CARBAMAZEPINE LEV" and "CARBAMAZEPINE MEASUREMENT", there is a significant cost difference depending on whether or not a word like "MEASUREMENT" can or should be deleted at some cost lower than the cost of transforming "LEV" to "MEASUREMENT" minus the cost of deleting "LEV". If it is possible, then the total edit cost would be the cost of deleting "MEASUREMENT" and deleting "LEV". The edit distance program uses the substitution table to determine whether or not such a deletion is worthwhile.

Word deletion is not the only useful rule. There is yet another option. If "MEASUREMENT" and "LEV" are synonyms, then they can be entered into the substitution table with a low cost. There are many ways to use general substitution. In the medical community, the Unified Medical Language System (UMLS) is a guide for standard abbreviations. When these abbreviations occur in hospital databases, they can be substituted with their corresponding definitions. For example, it is easy to see that "CSF (IGG)" and "Cerebrospinal fluid immunoglobulin G" are exactly the same. But, ignoring case differences and punctuation, it would take 29 character deletions to transform one string into the other. However, given that "CSF" can be

substituted for "Cerebrospinal fluid" and "IGG" for "immunoglobulin G" at low edit cost, a low total edit distance can be achieved. For other applications, related dictionaries can be used. Also, Roman numerals can be substituted for their Arabic counterparts and so on. These types of substitutions should all be included in the substitution table.

4. LEARNING ALGORITHM

String edit costs must be learned because fixed initial costs based on intuitive guessing lead to very low accuracy (approximately 40% for the data set below when we tried). The obvious learning approach is to use gradient descent in the total cost on a training set. This, however, has two major problems. In order to produce an optimal set of costs for the edit rules with the greedy algorithm, the cost for each rule is individually and sequentially optimized. This means that the cost for rule one is optimized followed by rule two and so on. Unfortunately, edit costs interact. The optimal cost for rule one will change depending on the optimal cost for rule two, so after producing a cost for rule two, the cost for rule one needs to be recalculated.

The interaction of rule costs has the consequence that the final costs derived by gradient descent are heavily dependent on initial values because there are several local maxima to which they can converge. The initial values determine final values and there is no way to pick good initial values other than extensive random restart. Random restart could, in theory, be used, but is requires significant computation. We use a modification of the genetic algorithm to avoid the convergence to local minima. The algorithm uses a set of genes (20 in the example below), each of which represents an individual cost set including all of the edit rules. First, each gene is initialized with some random set of costs. Next, each gene's fitness is calculated.

To calculate the fitness of a gene, the gene is given a set of N string pairs which are labeled as to whether or not they refer to the same item. (We use N=100 below.) This comprises the training data. The gene predicts whether or not a pair of strings refer to the same item by finding the edit distance using its own costs and comparing the result with the threshold. It keeps a total of how many pairs it has guessed correctly. If the gene guesses correctly, its fitness is increased by one. If the gene guesses incorrectly, its fitness is decreased by the difference between its guess and the necessary threshold distance, which we set to 10% the length of the longer string. This favors better genes, even if they do not guess correctly.

The genes are then sorted based on their fitness. The nine genes with the best values are reproduced. The best fitness is reproduced four times and all the other genes are reproduced twice. Finally, the genes' costs are mutated randomly. Costs converge to optimal values after several generations of genes. This method solves the local maxima problem because the edit costs are calculated simultaneously through gene reproduction and mutation. We have not attempted to tune the many parameters in the genetic algorithm; fortunately, it worked well on the first parameters selected. The algorithm converges to stable values after approximately 100 generations.

5. RESULTS

We use a set of 100 string pairs as training data to learn the string edit costs associated with each edit rule. For our data set, we use pairs labeled with costs of "1" for difference and "0" for similarity, but approximate matches could equally easily be used. We had a set of strings labeled by physicians as referring to the same item, or as probably not referring to the same item. For example,

SAME

atropine sol oph 1% 5 ml
atropine sulfate sterile ophthalmic solution

stat liver profile
organ or system related test nos

augmentin 250mg/5ml
brl 25000

absorb gelat oph film 25x
gelatin sponge absorbable

DIFFERENT

stat creat kinase
creatine kinase measurement

stat creatinine
creatinine measurement serum

stat digoxin
digoxin measurement

stat ggt
gamma glutamyl transferase measurement

Note that the "different" strings are near misses; they are sufficiently similar that it is unclear whether the physicians were correct or not.

Our algorithm also makes use of a substitution table of word and phrase substitutions and deletions. The substitution table is a list of string pairs that have exactly the same meaning, so one string can be substituted for the other with no loss in accuracy. A few typical substitutions are:

hep
hepatitis

ggt
gamma glutamyl transferase

cath
catheter

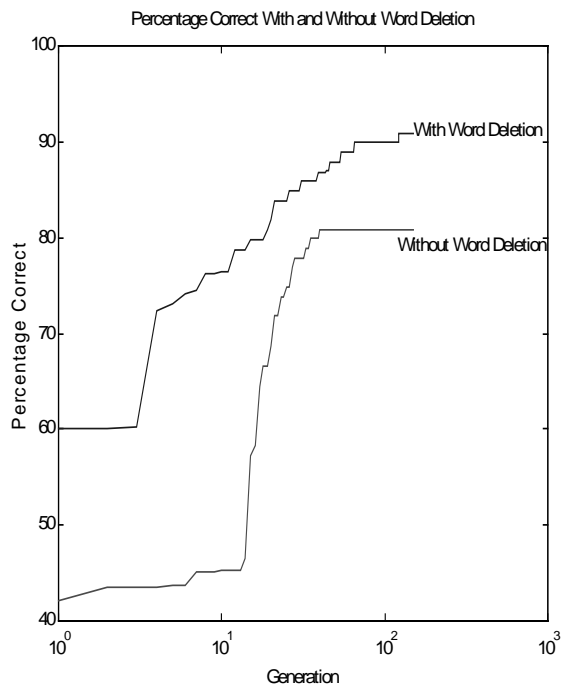
Substituting a word or phrase with the null string is the same as word or phrase deletion. Words for which we substitute the null string (i.e., words we delete) include:

measurement
assay
test

In the results presented below, substitution costs are simply set to zero; one could easily optimize them as part of the learning procedure.

The sample data used is stored in table format in a text file. The entire file is preprocessed by a PERL script which changes all letters to lower case and removes all punctuation except the ':', '%', and '-' characters. It also puts each string pair into a format that can be read by the edit distance program.

Three sample sets of data, each consisting of 100 string pairs were used to test the code. Fifty string pairs have been predetermined to be similar and 50 are not. The two databases used are slightly asymmetrical because one database contains more abbreviated terms than the other. For example, "antihuman globulin test coombs test indirect titer each antiserum" is abbreviated to "indirect coombs". Pairs were selected so that there was not a large difference between the lengths of the two strings. However, the asymmetry does lead to the low cost for repeated character deletions produced by the learning program. Learned costs for the sample training data was shown in Table 1. In these sample data sets, words such as "measurement", "test", "assay", and "screen" appear often. These words were added to the substitution table.



Graph 1: Comparison of Percentage Accuracy for Algorithm With and Without Word Deletion

6. CONCLUSIONS

Using dynamic programming to calculate string edit distances provides a powerful approach to determining similarity of items described by the strings. Dynamic programming is much more efficient than alternative methods that involve evaluating the total edit cost for each permutation of edit rules. It is easy to add new rules to this set to improve performance.

One key set of rules to add is word and phrase substitution and deletion. Transforming one string into another is traditionally used for spell checking, but this simple case only involves inserting, deleting, or transposing letters. Adding word substitution allows one to correctly match "Hep A" to "Hepatitis A" rather than to "Hep B" if there is a rule that "Hepatitis" can be abbreviated to "Hep". (Depending on the costs learned for multiple deletion, one might also learn that deleting the end of the word "Hepatitis" costs less than changing the single letter. In the medical domain, three letter abbreviations are frequent.) In our test runs, adding a simple table of word and phrase deletion and substitution rules improves the accuracy of the program by approximately 10% with.

On average, over three data sets, the percentage accuracy increases when word deletion is added. The method with word deletion starts out at 58% accuracy whereas the method without word deletion starts at 40% accuracy. After 150 generations, the values converge to 90% and 80% accuracy respectively. These trends are shown in Graph 1. Performance is, of course, dependent on the data set and the frequency of the deleted words in it.

We learn edit costs using a genetic algorithm, thus avoiding the local minima and slow convergence problems found using gradient descent. It is possible to gain a high degree of accuracy of 80% to 90% by training the program on a small data set. Learning the costs of different rules also allows us to use rules or substitutions which we are unsure of; if they prove to be bad, then they are given high cost.

We have presented this work in the context of the merge/purge problem for databases. The notion of flexible matching of strings is, however, much more widely applicable. Often one has descriptions of items in free or semi-structured text: for example, product descriptions extracted from the web or from databases. These descriptions contain many words that are either synonyms (e.g. "cost" and "price" or "Palm" and "Palm Pilot") or words that should probably be deleted (e.g., "excellent" or "system"). Our enhanced string edit function can account for these differences.

7 ACKNOWLEDGEMENTS

We thank Andrew McCallum and Kamal Nigam for the initial discussions which inspired this work.

8. REFERENCES

- [1] Hernandez, M.A.. and Stolfo, S.J. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD*, 127-138 (1995).
- [2] Hylton, J. Identifying and merging related bibliographic records. MIT LCS Masters Thesis (1996).
- [3] Kilss, B., and Alvey, W. (Eds.) *Record Linkage Techniques - 1985*. Statistics of Income Division, Internal Revenue Service Publication 1299-2-96. Available from <http://www.fcsm.gov/> (1985).
- [4] McCallum, A, Nigam, K., and Ungar, L.H.) Efficient Clustering of High-Dimensional Data Sets with Applications to Reference Matching. *Proceedings of the KDD-2000*. (2000).
- [5] Monge, A..E., and Elkan, C.P. The field-matching problem: algorithm and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 267-270 (1996).
- [6] Newcombe, H. B., Kennedy, J. M., Axford, S. J. and James, A. P. Automatic linkage of vital records. *Science*, 130, 954-959 (1959)
- [7] Ristad, E. S., and Yianilos, P.N. Learning string edit distance. Research Report CS-TR-532-96, Department of Computer Science, Princeton University, Princeton, NJ, October 1996, Revised (October, 1997).