RESEARCH REPORT SERIES
(*Statistics #2004-02)*

An Adaptive String Comparator for Record Linkage

William E. Yancey

Statistical Research Division
U.S. Bureau of the Census
Washington D.C. 20233

Report Issued: February 19, 2004

An Adaptive String Comparator for Record Linkage*

William E. Yancey
U.S. Bureau of the Census

March 4, 2004

Abstract

We develop a string comparator based on edit distance that uses variable edit-step costs derived from training data. Using first and last name data from Census files, we compare the performance of this string comparator with one without variable edit step costs and with the Jaro-Winkler string comparator, which is standardly used in the Census Bureau's record linkage software.

1 Introduction

A string comparator is a function that returns a numerical comparison value for a pair of strings. Specifically, if Σ is an alphabet of characters and Σ^* is the set of strings (finite character sequences) from this alphabet, then a string comparator is a function c where

$$c : \Sigma^* \times \Sigma^* \to \mathbb{R}.$$

One uses different string comparators for different purposes. For example, the C computer language utility $\mathsf{strcmp}(s_1, s_2)$ applied to strings s_1, s_2 returns an integer whose absolute value is the position of the first character pair that disagrees and whose sign is given by the lexicographical order of these disagreeing characters. This string comparator is useful for sorting a set of strings into lexicographical order or for searching for a given string in a set of sorted strings. The study of *approximate string matching* is generally directed toward a somewhat different application. Instead of searching a set of strings for an exact match of a given key string, one wants to search a set of strings for those strings that "nearly" match a given string. In this case, one wants a string comparator that is a metric that computed the "distance" between two strings, so that

*This report is released to inform interested parties of ongoing research and to encourage discussion of work in progress. The views expressed are those of the authors and not necessarily those of the U. S. Census Bureau.

one can retrieve all of the strings within a given distance of the key string. A candidate for such a metric is *edit distance*, which is discussed below in Section 3.2.

For record linkage, the application is somewhat different. Instead of a searching task, we have a decision problem. Given two strings, we must decide whether they agree, where by "agree" we really mean that the two strings were both intended to represent the same word. Of course it is not going to be possible for the algorithm to determine intent. We want a number that represents a degree of similarity between the strings where increasing similarity will be interpreted as our increasing confidence that the two strings both represent the same underlying entity.

The Census Bureau's record linkage software has a string comparator that is used for probabilistic record linkage. Below we develop another string comparator based on edit distance which can be trained to adapt its evaluations to a body of data. Below we describe the record linkage application for the string comparator, then we describe the current Census Jaro-Winkler string comparator, then we describe a string comparator based on edit distance, and then we describe the adaptive version of this string comparator. We then compare the performance of these string comparators applied to the record linkage application.

2 Record Linkage Application

For Fellegi-Sunter record linkage theory, under the conditional independence assumption, the comparison weight of two records is computed as the sum of the comparison weights of the individual comparison field values. For a given comparison field, the agreement weight is computed as

$$a_w = \log \frac{\Pr(\gamma = 1 \,|M)}{\Pr(\gamma = 1 \,|U)}$$

and the disagreement weight is given by

$$d_w = \log \frac{\Pr(\gamma = 0 \,|M)}{\Pr(\gamma = 0 \,|U)}$$

where $\gamma = 1$ indicates that the field values of the record pair agree and $\gamma = 0$ indicates that they disagree. The probabilities are conditioned on M, the two records are a match (*i.e.*in truth represent the same entity) and U, the two records are not a match. The agreement probabilities $\Pr(\gamma = 1 \,|M)$ and $\Pr(\gamma = 1 \,|U)$ are given as input parameters to the record linkage procedure. The assignment of either agreement weight or disagreement weight is straightforward if, for instance, the given comparison field represents a categorical variable such as sex or race, but when the field contains strings, the linkage can be more robust if we allow more than two possible results. Suppose we consider γ to be the value of a string comparator that takes the value 1 when the

two strings are identical and the minimum value 0 when the strings are totally different. We can then assign a field comparison weight to a string field pair of

$$w(x) = \log \frac{\Pr(\gamma = x \,|M)}{\Pr(\gamma = x \,|U)} \tag{1}$$

where $w(1) = a_w$, $w(0) = d_w$, and $w(x)$ is an increasing function on $[0, 1]$. If the string comparator value effectively reflects our level of confidence that the two strings both represent the same underlying word, then an approximation of w can produce a comparison weight that can interpolate between full agreement and full disagreement weights and quantitatively reflect the level of likelihood that the field values are a match.

We will next present three candidate string comparators. Later we will compare their results from applying them to real data to try to determine to what extent we are confident that we can use them to compute comparison weights for record linkage.

3 String Comparator Functions

3.1 Jaro-Winkler String Comparator

3.1.1 The Basic Comparator

The Jaro-Winkler string comparator [Winkler] is the comparator developed at the U.S. Census Bureau and used in the Census Bureau record linkage software. The basis of this comparator is the count of common characters between the strings, where a character is counted as common if it occurs in the other string within a position distance that depends on the string length. That is, suppose we are comparing the two strings

$$\alpha = (a_1, a_2, \ldots, a_m)$$
$$\beta = (b_1, b_2, \ldots, b_n)$$

where $m \leq n$. The search range distance d is defined to be

$$d = \left\lfloor \frac{n}{2} \right\rfloor - 1,$$

less than half the length of the longer string. Generally speaking, for $1 \leq i \leq m$, the character a_i will count as common with the character b_j when $a_i = b_j$, provided

$$i - d \leq j \leq i + d$$

and

$$1 \leq j \leq n.$$

Characters are counted as common only once, so that (x, a, b, c) and (x, x, w, y, z) have just one common character. For example, with (b, a, r, n, e, s) and (a, n, d, e, r, s, o, n),

we have $d = 3$, so that a, r, n, e, s are common for a count of 5 common characters.

The value of the string comparator is further determined by a count of transpositions. Transpositions are determined by pairs of common characters out of order. The above example would count one transposition since the corresponding common characters $(r, n), (n, e), (e, r)$ are out of order, so these three contain one pair. If c is the common character count and t is the number of transpositions of the strings α, β, then the basic Jaro string comparator score s_J is computed by

$$s_J = \frac{1}{3}\left(\frac{c}{m} + \frac{c}{n} + \frac{c-t}{c}\right).$$

We can see that if the strings are identical, then we have $m = n = c$ and $t = 0$, so that $s = 1$. If the strings are not identical, then we must have either $c < m$ or $c < n$ or $t > 0$, so that $s < 1$. For all compared pairs of strings, we have $s \geq 0$. For the above example, the basic score is

$$s_J = \frac{1}{3}\left(\frac{5}{6} + \frac{5}{8} + \frac{4}{5}\right) = \frac{271}{360} \doteq 0.7528.$$

A further enhancement to the score computation due to Winkler is based on the observation that typographical errors occur more commonly toward the end of a string, so that strings that agree on prefixes (1 to 4 characters) are given a higher comparison score. Specifically, the algorithm considers the first four prefixes of each string and lets p be the length of the longest prefix pair that agree exactly ($a_i = b_i$). The adjusted Winkler comparison score s_W is given by

$$s_W = s_J + \frac{p\,(1 - s_J)}{10}.$$

Note that this formula preserves the property that $0 \leq s_W \leq 1$ and $s_W = 1 \Leftrightarrow s_J = 1 \Leftrightarrow \alpha = \beta$. In practice, this score adjustment is only applied to pairs of strings that are reasonably similar, specifically $s_J > 0.7$. In this example, since the strings differ in the first character, none of the prefixes agree, so $p = 0$, and there is no change in the score.

3.1.2 The Adjusted Comparator

There are two relatively recent enhancements to the Jaro-Winkler string comparator that are currently used by default, but can be optionally omitted. One option is intended to augment the score for some longer strings. It is applied if the strings are sufficiently long and the substrings beyond the common prefix have enough common characters and a high enough proportion of common characters. If the string pair meets the requirements, the adjustment can only increase the score.

The other adjustment is designed to account for commonly occurring character subtitutions that occur in typographical error. The program has a list

of pairs of similar characters, which represent common typographical substitutions, based on common spelling mistakes, visual similarity, or keyboard proximity. The current version uses 36 such pairs. After the common characters have been matched, the program looks through the unmatched characters to see if an unmatched character in one string is similar to an unmatched character in the other string, allowing character in the other string to be similar to at most one character from the first string. If s is the total number of similar character pairs counted, then the adjusted count c_s of common characters is given by

$$c_s = c + 0.3s$$

and if this adjustment is in force then the basic Jaro score s_J is actually computed by

$$s_J = \frac{1}{3}\left(\frac{c_s}{m} + \frac{c_s}{n} + \frac{c-t}{c}\right).$$

In our example, the only unmatched character in the first string is b, and the only characters similar to b in the list of similar characters are v and 8, which do not occur in the second string, so no similar character adjustment is made.

3.2 Edit Distance String Comparator

Another approach to string comparison is based on edit (or Levinshtein) distance. These measures are determined by the minimum number of edit steps required to convert one string to the other. We will call *edit distance* the method that uses insertion, deletion, or substitution for possible edit steps. Another possibility is to restrict the edit steps to just insertion or deletion. The minimum number of edit steps can be computed using a straightforward dynamic programming algorithm with complexity $O\left(n^2\right)$, where n is the length of the strings. The idea of the dynamic programming algorithm is to compute the shortest distance for all prefix pairs, where we compute the cost of the current prefix pair from the minimum of the previous minimal prefix costs plus the cost of the edit step that converts the earlier prefix to the current one. Since the strings representing Census names are fairly short (generally less than 20 characters), the algorithm executes very quickly. These distance measures have the property that they are metrics. It is clear that the distance functions are reflexive and symmetric, and the triangle inequality can be verified by induction on the length of the strings. In our studies, we have used edit distance since this seems to capture an appropriate edit process. Many misspellings involve substituting one letter for another.

In order to use one of these distance functions as a string comparator and to compare its performance to the Jaro-Winkler string comparator, we see from (1) that it is helpful to convert the distance value to a number between 0 and 1, with 1 representing complete agreement. The resulting function may be called a *similarity function* [Navarro]. We do this by considering the maximum number of edit steps possible to convert one string to another. For edit distance, if the strings of length m and n with $m \leq n$ have no characters in common, then the

minimum edit sequence is to do m substitutions followed by $n-m$ insertions or deletions, for a total of n edit steps. Thus we can scale edit distance into our $[0,1]$ comparison measure by letting the comparison value x be given by

$$x = 1 - \frac{d}{n}$$

where d is the edit distance between the two strings and n is their maximum length.

When first experimenting with this edit similarity function, we found that it could be too severe, penalizing a string pair for every difference without giving sufficient credit for common features. For example, the pair {Stan, Stanley} has an adjusted Jaro-Winkler score of 0.9142 while this edit distance similarity function evaluates the pair at 0.5714. We decided to modify the edit distance similarity score by taking into consideration the length of the *longest common subsequence* (lcs) of the two strings. This length can be computed similarly as the edit distance by counting the number of identity steps (no edit required) in the shortest edit sequence. However, this is only necessarily true when the available edit steps are restricted to insertion and deletion [Wagner]. This can be done with a separate but parallel distance computation. Since the maximum possible length of an lcs is m, the length of the shorter string, we can get an lcs similarity score y from

$$y = \frac{l}{m}$$

where l is the length of an lcs. We derive a modified edit distance similarity score s_{edt} taking the average of these two scores

$$s_{edt} = \frac{x+y}{2}.$$

We note that the modified edit distance score for {Stan,Stanley} is 0.7857.

3.3 Adaptive String Comparator

The edit distance metric counts the minimum number of character edits required to convert one string to another, each edit having a unit cost. We consider the problem of having the edit cost function, instead of being constant, depend on the characters involved. If we are trying to decide whether two strings both represent the same word, then edit changes that represent common misspellings or typographical errors could cost less than rare changes, so that the score might better reflect the plausibility that the two strings are equivalent. Sources of typographical error could be written misspelling, phonetic misspelling, key stroke errors, or scanning errors. Since different data sets might be subject to different proportions of these error sources, it might be helpful if the edit cost function could adapt to the data sets at hand. We describe a method for assigning these costs and define an adaptive string comparator based on the edit comparator model.

But first we should be aware that there are some theoretical drawbacks involved in any string distance function that varies with the characters. The main one is that the resulting "distance" function may not be a metric. It is reflexive and the cost function can be defined to be symmetric, but the triangle inequality may fail. For example, for three one character strings, $\alpha = (a), \beta = (b), \gamma = (g)$, it may be that the cost function

$$c : \bar{A} \times \bar{A} \to \mathbb{R}^+$$

where $\bar{A}$ is the character alphabet (including the null character), could have

$$c(a,b) + c(b,g) < c(a,g)$$

if (a,b) and (b,g) happen to be common substitutions while (a,g) is a rare substitution.

In order to compute the variable costs, we use a probability model to compute cost values based on maximum likelihood. For characters $a_i, a_j \in \bar{A}$, let

$$p_{ij} = \Pr((a_i, a_j) | M),$$

the probability that the edit step (a_i, a_j) is used in a minimum cost edit sequence converting a pair of matched strings. A connection between cost and probability (frequency) can reasonably be modeled as

$$c(a,b) = -\kappa \log \Pr((a,b) | M)$$

where by $\Pr((a,b) | M)$ we mean that, given an edit has occurred in converting matched strings, the probability that it was the edit $(a,b), a, b \in \bar{A}$. We introduce a scale factor $\kappa > 0$ because there does not appear to be any analytic reason determining the use of the natural logarithm. We may wish to choose a base that produces a conveniently scaled cost function. For example, suppose that the alphabet A has n characters and we a trying to estimate the costs for $\frac{1}{2}n(n+1)$ different edit steps. We initially would probably set all of the edit steps to have equal cost, so that initially we have

$$c(a,b) = -\kappa \log \frac{2}{n(n+1)}.$$

Thus it might be convenient to let

$$\kappa = -\frac{1}{\log \frac{2}{n(n+1)}}$$

so that all initial costs are set to equal 1.

To estimate these probabilities, we can use a set of training data consisting of pairs of strings that are likely matches. Using this data, we can compute the maximum likelihood values for these probabilities using an EM algorithm approach.

3.3.1 Maximum Likelihood

For each edit step $(a_i, b_j) \in S \subset C^* \times C^*$, we want to maximize

$$L = \prod_{i,j} \Pr\left((a_i, b_j) | \, M \right)^{n_{ij}}$$

where the data n_{ij} is the count of times the edit step was used and the parameters are $\Pr\left((a_i, b_j) | \, M \right)$, which is the probability of using the edit step (a_i, b_j) in a minimum cost edit of a pair of strings which both represent the same word. For a given count of the edit steps, the maximum likelihood occurs for

$$\Pr\left((a_i, b_j) | \, M \right) = \frac{n_{ij}}{N} \tag{2}$$

where N is the total of all edit steps used. That is, we count the edit steps used in the least cost edit path for all pairs of likely matching strings. Ristad [Ristad] also uses a probabilistic approach, but counts all edit step through all possible paths. Using the least cost path seemed to be the more reasonable model for this application.

Using the EM algorithm to maximize likelihood, we can initialize the parameter values with equal probabilities; for the E-step, we can count the number n_{ij} of each edit steps used; for the M-step, we use the edit frequencies to revise the edit probability parameters according to (2).

3.3.2 Developing a Similarity Score

While the above calculation is simple and converges quickly, the resulting costs may lack some rationality. For example, for a given pair of characters (a, b), we may get a cost function c where

$$c\,(a, b) > c\,(a, \varepsilon) + c\,(\varepsilon, b)$$

that is, the cost of substitution can exceed cost of the equivalent insertion and deletion. We modify the cost function by replacing the substitution cost by the minimum of these two. For a given pair of strings, we can then use the dynamic programming algorithm to compute their adaptive edit "distance". To convert this to a comparison score, we still need to divide by the maximum distance. The concept of maximum distance between two unrelated strings is less clear. Instead of simply counting the maximum number of edit steps, we have to decide on what costs to assign to each step. By analogy to the edit step count, we compute a maximum distance m_a by

$$m_a = \sum_{k=1}^{m} c'\,(a_k, b_k) + \sum_{k=m+1}^{n} c\,(b_k, \epsilon)$$

where

$$c'\,(a, b) = \begin{cases} c\,(a, b) & \text{if } a \neq b \\ 1 & \text{if } a = b \end{cases}$$

If c_a is the total minimum adaptive edit cost, we compute an adaptive similarity score s_{adp} by

$$s_{adp} = \frac{1}{2}\left(\frac{c_a}{m_a} + \frac{l}{m}\right)$$

where l is the lcs length as before.

3.3.3 The Training Data Set

In order to calculate our maximum likelihood costs, we need a set of training data. To produce a training data set, we run the matching software on the test files and get a printout of matched record pairs sorted by match score. As with standard matching procedure, we examine the list and choose a cutoff score above which all the pairs are designated matches. We then read in the pairs of designated matched records and write out the pairs of first or last names where neither name is blank and the names are not identical strings. This forms the initial training data set of pairs of names that come from matched records but have some spelling discrepancy. We want pairs of strings that likely both represent the same name. This file is edited further to eliminate the most unlikely agreeing pairs. We read in these name pairs and compute the standard Jaro-Winkler string comparator score. We then sort the pairs by this score and eliminate low scoring pairs that do not appear to be different spellings of the same name. The result is a set that represents pairs of strings that are matched but which have spelling errors. This procedure could be used to produce training data from any pair of files designated for record linkage.

4 Testing the String Comparators

Our test data consists of the three 1990 Census/PES file pairs that have been clerically reviewed. So far we have used the largest set (STL) to explore the methodology for our comparison tests. In the future we can apply the same methods to the other two sets to see if the results are consistent. We use the Census Bureau matcher software to produce our training data and test data. The standard blocking strategy for these sets has been to use the cluster number and the first character of the last name. We used this to produce the first name data, but it presumably introduces a bias for the last name data, so we reran the software blocking on cluster number and first character of first name to produce the last name data.

Specifically, for each of these two runs, we ran the counting program and used the output of the EM algorithm to obtain parameter estimates for the agreement probabilities. Then we used these parameter estimates to run the matching program. As described above, we used the output of the matcher to produce training data sets. For the test data, we considered every record pair brought together by the blocking criterion. When blocking on last name character, we printed out every pair of first names from these record pairs. Since we had clerically reviewed truth data, we printed these first name pairs

to two different files, depending on whether the pair came from a match pair or a non-match pair. We did the analogous thing for last names under the other blocking criterion.

4.1 Linear Regression

Although we have scaled all of the string comparator functions to produce scores between 0 and 1, these scores are not modeled as probabilities. The specific scores have no particular significance. It does not make much sense to compare the scores of two string comparators directly. However, increasing comparator scores are supposed to indicate increasing confidence of string pair matching. Thus valid string comparators should exhibit a common trend. Thus we compare two comparators by computing a linear regression of the scores of one onto the other.

We computed separate regressions for first name pairs and last name pairs. In both cases, we used the set of pairs that came from the designated match records. We reduced the sets by eliminating all identical string pairs and all repetitions of the same string pairs to arrive at a set of unique, unequal string pairs. We then computed residuals for both the sets from match pairs and from non-match pairs.

4.1.1 The Jaro-Winkler String Comparators

We compared the results of the Jaro-Winkler string comparators, with and without the two adjustments for long strings and similar characters. We did this partially to try to perceive the effects of the adjustments and partially to get a baseline for similar string comparators.

Not surprisingly, the two J-W versions are highly correlated (0.99) and the linear regressions are a tight fit, the last name pairs slightly better than the first name pairs (first $R^2 = 0.98$, last $R^2 = 0.99$). The first name pairs have very few residuals greater than 0.1, the last name pairs have none (first 99.8%, last 100.0% have $|r| < 0.1$).

4.1.2 Edit Distance and Adaptive String Comparators

When we compare the edit distance and adaptive string comparators, the results are very similar. The are similarly correlated (0.99) and the regression fit is about as good (first $R^2 = 0.97$, last 0.99). There are only slightly more residuals larger than 0.1 (first 99.4%, last 98.8% have $|r| < 0.1$).

4.1.3 The Jaro-Winkler and Edit Distance String Comparators

We compared the adjusted Jaro-Winkler string comparator to both the edit distance string comparator and the adaptive string comparator. Although these still show strong similarities, they are not as similar as the previous comparisons. The J-W and edit distance comparators are strongly correlated (first 0.93, last 0.95) and have good regression fits (first $R^2 = 0.86$, last $R^2 = 0.90$). The

number of larger residuals is greater (first 91.9%, last 90.0% have $|r| < 0.1$). The J-W and adaptive string comparators are almost as correlated (first 0.92, last 0.94) and slightly less good regression fits (first $R^2 = 0.85$, last $R^2 = 0.89$) with slightly fewer large residual pairs (first 92.8%, last 90.5% have $|r| < 0.1$).

4.2 Comparison Weight Function

We have so far compared the string comparator functions by regressing one of their scores on the other, observing how much one set of scores explains the other, and examining cases where the actual and predicted score most differ. We note that for pairs where the scores were low, a larger residual probably did not matter for the record linkage application. This is because in the record linkage application, we use the comparator scores to evaluate a comparison weight function

$$w(x) = \log \frac{\Pr(\gamma = x \,|M)}{\Pr(\gamma = x \,|U)}$$

to assign a comparison weight. To evaluate the performance of a string comparator for the record linkage application, we need to compare the results of the comparison weight function. In order to do this, we need an appropriate comparison weight function for each comparator.

As we see from its definition, the evaluation of the comparison weight function depends on probabilities that are conditioned on the true match status of the records. While this may not be known in general, we take advantage of our Census test decks with reviewed match status to approximate a reasonable comparison weight function for each comparator. For first and last names, we use the full sets of pairs from matches and pairs from non-matches, not removing duplicates or exact matches. We calculate the string comparator value for each pair and count the number of pairs whose values fall within each cell of a partition of $[0, 1]$. To summarize the results, we see by inspection that generally the cell values of $w(x)$ decrease as x decreases from 1, until x gets to a point c where the values of $w(x)$ tend to level out. This corresponds to our notion that string comparator scores below a certain level should indicate total disagreement. In the region $[c, 1]$ where $w(x)$ seems to be increasing, we fit a straight line. The final approximated comparison weight function is a straight line truncated below by the total disagreement weight and truncated above by the total agreement weight.

For this data set, we found that for all four comparators, the first names resulted in a better fit than the last names. The worst fitting cases suggest that a logistic curve might be a better fit. The regression lines for all of the J-W cases were similar and those of the edit and adaptive methods were all similar. The J-W slopes were greater than the edit or adaptive slopes. This is consistent with the score regressions, where edit or adaptive scores the predicted by the J-W scores are always larger than the original edit or adaptive scores on $[0, 1]$.

4.3 Matching Weight Comparison

We compare the performance of the string comparators by comparing the agreement weights assigned to string pairs, using the estimated comparison weight function for each comparator. We looked at the four sets of first and last name, match and non-match records separately. Again our sets are reduced to the unique, non-identical pairs. We compute the weight differences between pairs of comparators, the same pairs for which we computed regression lines.

Again, as we might anticipate from the regression results, on the whole, the differences are not great. To give a snapshot example, suppose we consider a substantial weight difference to be 1.75. For the first and last name agreement and disagreement weights, this represents roughly 25% of the length of the total weight interval. The following is a table shows the percentage of pairs in our data sets for which $[\Delta] < 1.75$, where Δ is the difference between the two weight assignments.

	First Mat	First Non	Last Mat	Last Non
Basic & Adjusted J-W	99.1	99.8	99.9	99.8
Edit & Adaptive	100.0	99.9	99.8	99.9
J-W & Edit	96.1	97.8	98.0	99.2
J-W & Adaptive	96.4	98.0	97.8	99.3

(3)

Table (3) indicates that the string comparators assign agreement weights that are fairly similar for the preponderance of the sample data pairs. The non-matches have a lot of agreement since most of the pairs have very little similarity and thus get assigned the full disagreement weight.

The weight difference distributions suggest very modest differences between the basic and adjusted J-W comparators, which is not too surprising since they are highly similar. If anything, the differences between the edit and adaptive comparators are even less. This calls into question the benefit of all of the extra work involved in calibrating and computing the variable edit weights. Although the results are similar, the edit and adaptive string comparators show somewhat greater differences with the adjusted J-W comparator. This reflects that they have different structures.

On the one hand this is reassuring that reasonably valid string comparators should all provide fairly consistent results to the record linkage algorithm. On the other hand, it does not help us to choose between them. The different comparators might significantly affect the record linkage results for the cases with the most extreme differences, so we considered these individual cases.

Subjectively there does not appear to be a strong case for favoring the results of any of these comparators over the others. In general when one looks at the extreme cases for a matched pair set, the comparator assigning the higher weight appears to be doing a better job. When one looks at the extreme cases from the non-match pairs, the comparator assigning the higher weight appears to be doing worse.

For example, when we look at the differences between the basic and adjusted J-W comparators, as the Table (3) suggests, there are not many cases of large changes. The changes that improve the matching are countered by similar changes that do not improve the matching. Likewise, comparing the edit and adaptive comparators, there are even fewer large changes. Among these, the comparator giving the better weight assignment is fairly evenly balanced. In these direct comparisons, there does not appear to be a case for using the character specific adjustments.

When we compare the adjusted J-W comparator to either the edit or adaptive comparator, we do see more sizable differences. These differences point out the distinct properties of the two types of string comparator, but they do not clearly establish which comparator is doing the better record linkage job. The basic difference is that the J-W string comparator is good for identifying pairs of strings with permuted common characters, while the edit-based comparators are good for finding pairs of strings with significant common substrings.

To cite some specific examples, there are cases where the J-W comparator assigns the full disagreement weight and the edit and adaptive weight are more than 5 points higher, resulting in a strong agreement weight. The J-W comparator is exacting on short strings, so that pairs such as {UZ, U Z} and {JK, JIK} that receive full disagreement weight from J-W almost or more than 6 points higher weights from the edit and adaptive comparators respectively. Likewise, the substring property gives similarly strong weight increases to {Ivy, Ivory}, {Sara, Asara}, {Tony, Anthony}, {Dale, Lyndale}, and {Leroy, Roy}, which are subjectively credible as positive agreement weight pairs. On the other hand, these comparators give similar weight increases to {Amy, Tammy}, {Randy, Ray}, and {Tracy, Ray}, which appear to be less likely matched names. Likewise among last names, the edit and adaptive comparators give a weight increase of from 3 to 5 points to double names like {Shell Gladney, Gladney} or {Cohee-Wheeler, Wheeler}. On the other hand, there are also strong increases for {Edwards, Ward}, {Hermann, Mann}, and {Ore, Moore}.

The differences where the J-W weight exceeds the edit or adaptive weight are not as large as the largest of the above differences. On the positive side, the J-W exceeds the others by around 3 points for {Tolando, Talonda}, {Bilenda, Belinda}, and Louanda, Lavonda}, but also less convincingly for {Geraldine, Regna}, {Cornelius, Rocile}, and {Lenora, Veronica}. Actually there are smaller differences for the last name match pairs, and the largest difference last name non-match pairs appear to be awarding slightly positive agreement weight to unlikely agreement pairs. By just looking at the largest weight differences for last name pairs, the edit and adaptive comparators appear to perform a little better than the adjusted J-W comparator.

By just subjectively evaluating the comparators' performance on some examples, we can get an idea about how they perform differently, but it does not provide convincing evidence for evaluating their relative performance for the record linkage application. Of course, the relevant test for that would be to compare the effects of the different comparators on the linkage of the record pairs. Since many other factors are involved in the record linkage, it would be

difficult to isolate particular comparator effects. Since it is difficult to observe definitive evidence of superiority by examining the comparator results directly, it would be even harder to discern when masked by many other contributions.

5 Summary

We have length of the longest common substring along with edit distance to define a string similarity function to compare with the Jaro-Winkler string comparator. We create an adaptive variation on the edit distance cost function to adjust edit costs based on frequency of use. We compare the results of the different comparators on first and last name pairs from Census files. Regression analysis shows that they all have strong common linear trends. By developing a record linkage weight assignment function for each comparator, we can examine differences in weight assignments between the comparators. In both the Jaro-Winkler case and the edit/adaptive case, we do not detect any real advantage from using either adjustment for commonly used substitutions. While largely very similar, the Jaro-Winkler comparator does have some large weight differences with the edit or adaptive comparators. There is some slight indication that the edit approach is doing more good than harm, but this would have to be confirmed using other data sets.

References

[Navarro] Navarro, G. "A Guided Tour to Approximate String Matching." *ACM Computing Surveys.* Vol. 33, No. 1, March 2001, pp. 31–88.

[Ristad] Ristad, E. S. and Yianilos, P. N. "Learning String Edit Distance." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1998, 20(5):522–532.

[Wagner] Wagner, R. and Fisher, M. "The String to String Correction Problem." *Journal of the Association for Computing Machinery.* Vol. 21, No. 1, January 1974, pp.168–173.

[Winkler] Winkler, W. E. *The State of Record Linkage and Current Research Problems.* Statistics of Income Division, Internal Revenue Service Publication R99/04. Available from http://www.census.gov/srd/www/byname.html.

A Appendix

A.1 Linear Regression

We provide a sample of the linear regression graphs of the scores from one string comparator onto the scores of another. In Fig (1) we see a strong linear relation

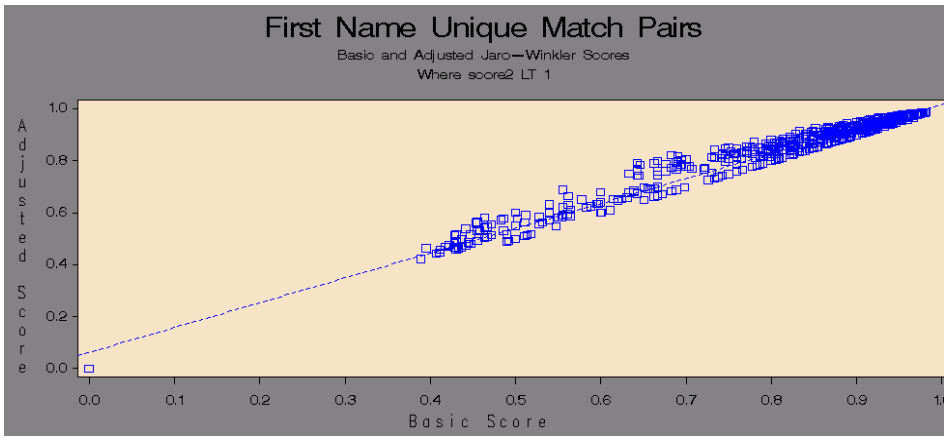

Figure 1: Basic and Adjusted Jaro-Winkler Scores

between the Jaro-Winkler scores with and without the adjustments for similar characters and longer strings.

In Fig (2), the edit scores with and without adjusted costs also show a strong linear relation. Several of the more outlying points have scores low enough that they likely both receive full disagreement weight.

In Fig (3), the adjusted Jaro-Winkler and adaptive edit scores show a dominant linear relationship, but there are more outliers and some of these represent scores high enough to result in some significant weight differences.

A.2 Comparison Weight Function

In Fig (4) we have plotted the binned comparison weights adjusted Jaro-Winkler scores for first names. The weights appear to line up well for scores above 0.6. In Fig (5) we plot the regression line for these points.

In Fig (6) we plot the last name comparison weights for edit distance. The weights for scores above 0.4 are not as linearly aligned, but the linear regression in Fig (7) provides a fair approximation.

A.3 Matching Weight Comparison

As we have seen in Table (3), using the computed weight comparison functions, the different string comparators assign weights of small differences to the bulk of name pairs. To give a little idea of where the comparators differ the most, we give some tables representing the largest differences.

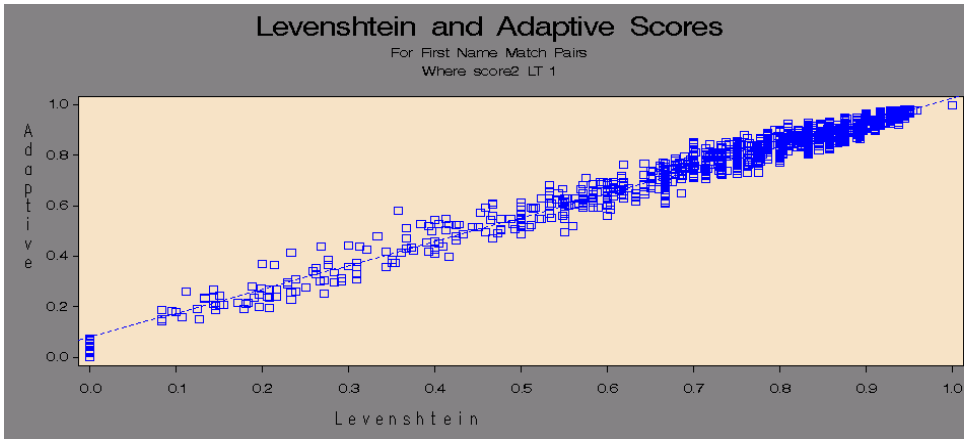

Figure 2: Edit and Adaptive Scores

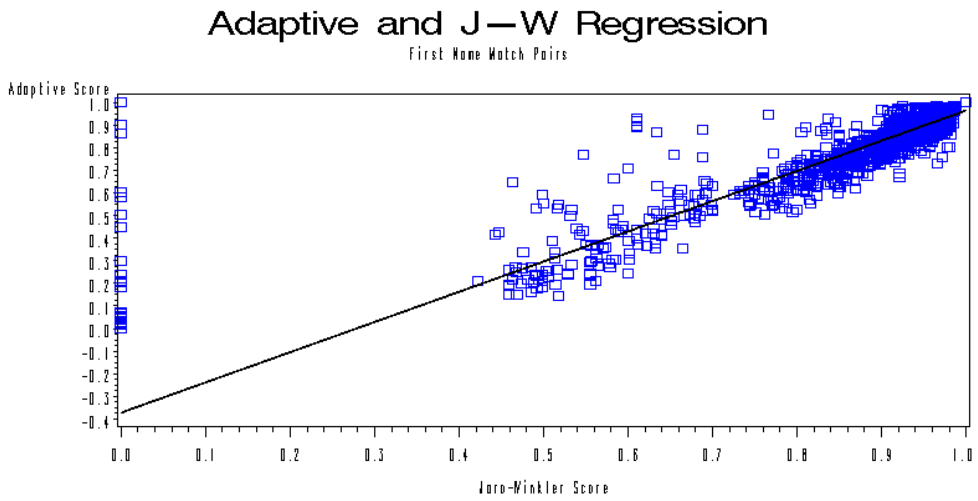

Figure 3: Adjusted Jaro-Winkler Score and Adaptive Edit Score

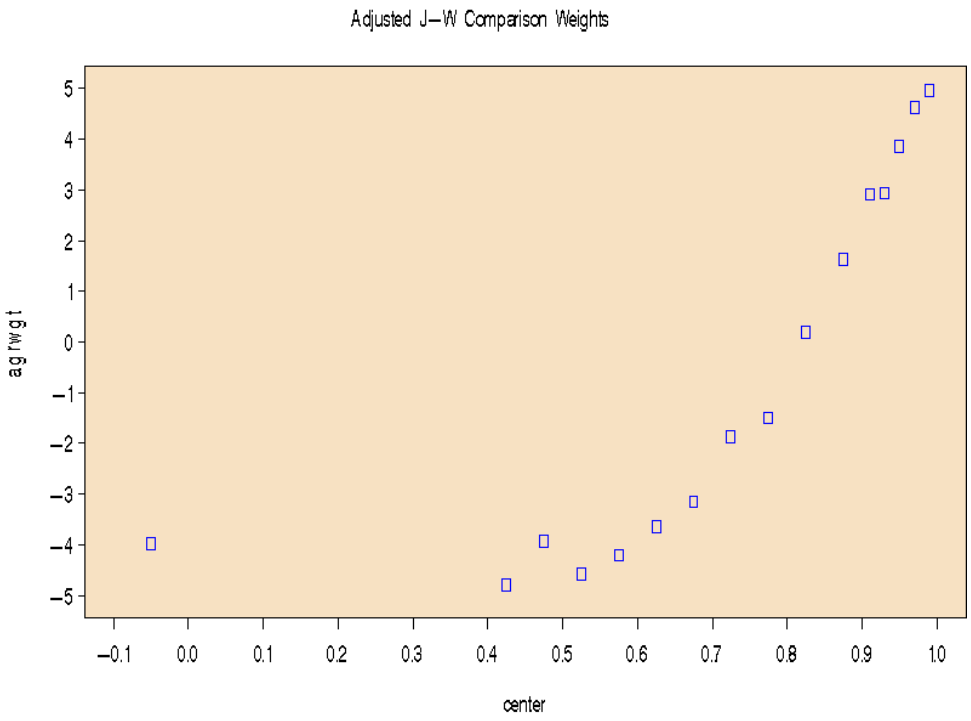

Figure 4: Adjusted Jaro-Winkler Comparison Weights

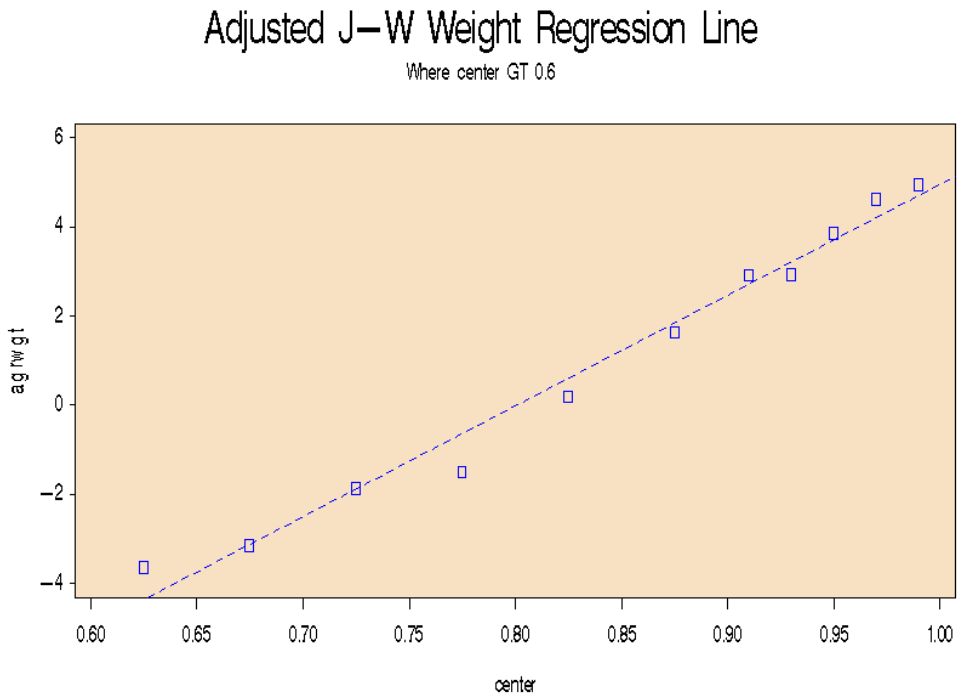

Figure 5: Adjusted Jaro-Winkler Weight Regression Line

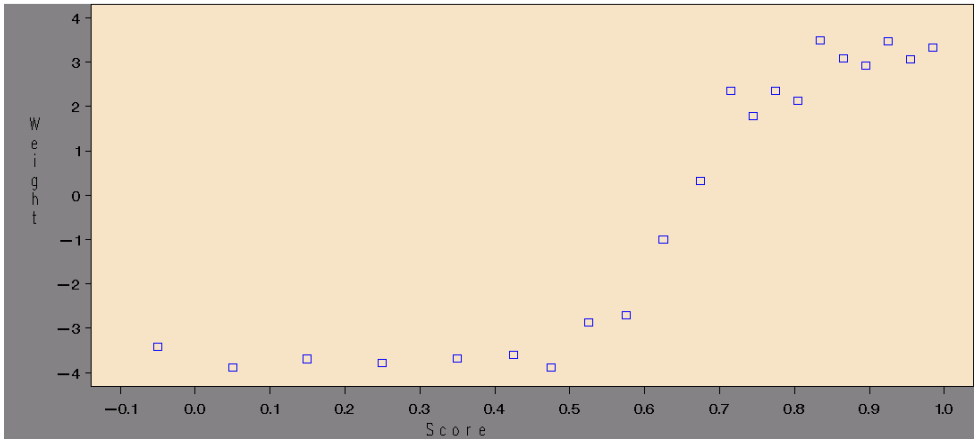

Figure 6: Edit Distance Comparison Weights

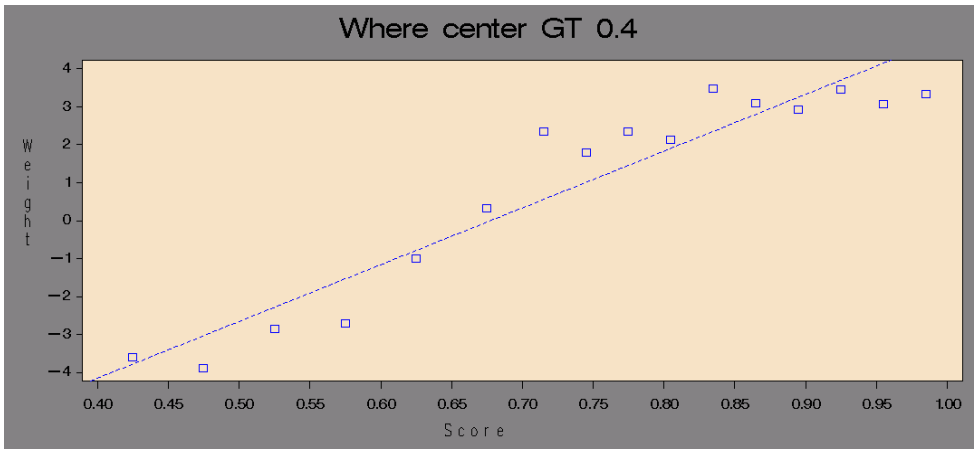

Figure 7: Edit Distance Weight Regression Line

A.3.1 The Jaro-Winkler String Comparators

In Table (4) we are looking at the last name pairs coming from matching records where the adjusted J-W weight most exceeds the basic J-W weight. Even the largest differences produced by the adjustments are not too dramatic and it is not clear that all of the pairs should have their comparison weight increased.

Adjusted Weight > Basic Weight

Last Name Match Pairs		J-W Scores		J-W Weights		
Name 1	Name 2	Basic	Adj	Basic	Adj	Diff
GRICE	GRIFFIN	0.6762	0.7973	−2.8908	−0.6981	2.1927
GORDON	JORDAN	0.7778	0.8778	−0.6127	1.1317	1.7444
HAMER	HAYMOND	0.6762	0.7684	−2.8908	−1.3566	1.534
JONES	JAMES	0.7600	0.8488	−1.0114	0.4726	1.4839
THOMPSON	JOHNSON	0.7131	0.8011	−2.0632	−0.6127	1.4504
CURRUTHERS	CORORRETHRS	0.7832	0.8697	−0.4915	0.9474	1.4389

(4)

In Table (5) we see the last name pairs coming from non-matching records where the adjusted J-W weight most exceeds the basic J-W weight. Here the largest differences arise from raising basic weights at or near full disagreement weight to adjusted weights that are still negative but nearer zero. Again it is

not clear that these adjustments would be helpful for the matching.

Adjusted Weight > Basic Weight

Last Name Non-Match Pairs		J-W Scores		J-W Weights		
Name 1	Name 2	Basic	Adj	Basic	Adj	Diff
MASEK	MASCARE	0.6762	0.8213	−2.8908	−0.1522	2.7386
GARANZINI	GARAVAGLIA	0.6852	0.8238	−2.6891	−0.0966	2.5925
THURMAN	THOMPSON	0.6905	0.8167	−2.5704	−0.2583	2.3121
MCCULLOUGH	MCCASKILL	0.6852	0.8092	−2.6891	−0.4285	2.2606
SCHMIDT	SCHUH	0.6762	0.7973	−2.8908	−0.6981	2.1927
COLEMAN	KOMADINA	0.6905	0.8113	−2.5704	−0.3810	2.1894
GRIFFIN	GREEN	0.6762	0.7958	−2.8908	−0.7327	2.1580
JACKSON	JAMES	0.6762	0.7958	−2.8908	−0.7327	2.1580

(5)

Since the J-W adjustments can only raise the basic score, the basic weight exceeds the adjusted weight most in cases where the adjustments had no effect and the scores remained the same. Tables (6) and (7) list the largest weight differences for match pairs and non-match pairs respectively. There a several other pairs with weight differences of size about 0.5. These differences should have small effect on the record linkage.

Basic Weight > Adjusted Weight

Last Name Match Pairs		J-W Scores		J-W Weights		
Name 1	Name 2	Basic	Adj	Basic	Adj	Diff
STUBBS	STUDES	0.8444	0.8444	0.8823	0.3735	−0.5088
BOYD	BOLDY	0.8267	0.8267	0.4836	−0.0309	−0.5145
DOWD	DOWELL	0.8250	0.8250	0.4462	−0.0688	−0.5150
VAUHNS	VAUGHER	0.8222	0.8222	0.3839	−0.1320	−0.5159
COAD	JACOAD	0.8056	0.8056	0.0102	−0.5110	−0.5213
TIMM	TZMMS	0.8050	0.8050	−0.0023	−0.5237	−0.5214
BUNCH	BUNTIT	0.7900	0.7900	−0.3386	−0.8649	−0.5263
HEE	YEE	0.7778	0.7778	−0.6127	−1.1429	−0.5302

(6)

Basic Weight > Adjusted Weight

Last Name Non-Match Pairs		J-W Scores		J-W Weights		
Name 1	Name 2	Basic	Adj	Basic	Adj	Diff
ALT	BALL	0.7222	0.7222	−1.8585	−2.4065	−0.5480
ORE	COLE	0.7222	0.7222	−1.8585	−2.4065	−0.5480
WARD	EDWARD	0.7222	0.7222	−1.8585	−2.4065	−0.5480
KEY	HUEY	0.7222	0.7222	−1.8585	−2.4065	−0.5480
OGE	LOVE	0.7222	0.7222	−1.8585	−2.4065	−0.5480
RICHARDSON	CARR	0.7167	0.7167	−1.9831	−2.5329	−0.5498
BILL	WILLIAMS	0.7083	0.7083	−2.1700	−2.7225	−0.5525
HALL	CHANDLER	0.7083	0.7083	−2.1700	−2.7225	−0.5525
MCDOWELL	COLE	0.7083	0.7083	−2.1700	−2.7225	−0.5525
HULL	FULLAURD	0.7083	0.7083	−2.1700	−2.7225	−0.5525
WILLIAMS	HILL	0.7083	0.7083	−2.1700	−2.7225	−0.5525
KERR	MAYBERRY	0.7083	0.7083	−2.1700	−2.7225	−0.5525

(7)

A.3.2 Adaptive and Edit Weight

The differences between edit weight and adaptive weight are generally smaller but more symmetric than between the two forms of J-W weight. For the last name pairs where the adaptive weight most exceeds the edit weight, we see that for the match pairs in Table (8) and non-match pairs in Table (9), the largest adaptive weight differences do not generally appear to benefit record linkage.

Adaptive Weight > Edit Weight

Last Name Match Pairs		Scores		Weights		
Name 1	Name 2	Adapt	Edit	Adapt	Edit	Diff
POLK	POKE	0.8600	0.7000	2.3178	0.3345	1.9833
YABER	YORBRO	0.6175	0.4667	−1.4263	−2.9108	1.4845
GANES	GAINS	0.8539	0.7333	2.2246	0.8332	1.3914
JONES	JOHNSON	0.7690	0.6476	0.9129	−0.4490	1.3619
YATES	YAT	0.8943	0.8000	2.8487	1.8304	1.0183

(8)

Adaptive Weight > Edit Weight

Last Name Non-Match Pairs		Scores		Weights		
Name 1	Name 2	Adapt	Edit	Adapt	Edit	Diff
BAKER	BLACK	0.7083	0.5000	−0.0242	−2.6572	2.6330
HARRIS	SHEARER	0.7598	0.5714	0.7716	−1.5887	2.3603
OWENS	BROWN	0.6735	0.4667	−0.5616	−2.9108	2.3492
BLASE	SIEBELS	0.6725	0.4381	−0.5763	−2.9108	2.3345
YATES	BRYANT	0.6699	0.4833	−0.6161	−2.9065	2.2904
PHILLIPS	HEMPHILL	0.7261	0.5417	0.2508	−2.0339	2.2847
AWLS	WELLS	0.7557	0.5750	0.7084	−1.5353	2.2437

(9)

In Tables (10) and (11) we have examples of last name pairs where the edit

weight most exceeds the adaptive weight. Again in the match pair and non-match pair cases, the differences are fairly small and do not appear to indicate generally better weight assignment for matching.

Edit Weight > Adaptive Weight

Last Name Match Pairs		Scores		Weights		
Name 1	Name 2	Adp	Edit	Adp	Edit	Diff
BLEIN	KLEIN	0.7616	0.8000	0.7993	1.8304	−1.0311
HAYES	BANES	0.5586	0.6000	−2.3359	−1.1613	−1.1746
WEST	WEBB	0.5570	0.6000	−2.3605	−1.1613	−1.1992
HEE	YEE	0.5819	0.6667	−1.9755	−0.1641	−1.8114

(10)

Edit Weight > Adaptive Weight

Last Name Non-Match Pairs		Scores		Weights		
Name 1	Name 2	Adp	Edit	Adp	Edit	Diff
PEPPERS	SELLERS	0.5058	0.5714	−2.9108	−1.5887	−1.3221
MAIER	LABER	0.5482	0.6000	−2.4963	−1.1613	−1.3350
REEDER	KREIGER	0.6122	0.6667	−1.5082	−0.1641	−1.3441
JORDAN	MORGAN	0.6095	0.6667	−1.5493	−0.1641	−1.3852
WILSON	DIXSON	0.6078	0.6667	−1.5764	−0.1641	−1.4123
REEDER	KRIEGER	0.5999	0.6667	−1.6983	−0.1641	−1.5342
WARNER	PARKER	0.5965	0.6667	−1.7498	−0.1641	−1.5857
TINKER	BINGER	0.5943	0.6667	−1.7836	−0.1641	−1.6195
TEIBER	BERNER	0.5283	0.6071	−2.8028	−1.0545	−1.7483
BOND	BOXX	0.4822	0.6000	−2.9108	−1.1613	−1.7495
PALMER	WALKER	0.5824	0.6667	−1.9678	−0.1641	−1.8037

(11)

A.3.3 J-W and Edit Weight

There are larger differences between either edit or adaptive weight and basic or adjusted Jaro-Winkler weight than are found between either pair of similar comparators. In Table (12) we see the largest weight differences for last names from matched pairs. Here we see one of the properties of edit score is that it gives credit for large common substrings even when they do not occur in the same parts of each string. For last names, this picks up some cases of double last names where the other string just has the second part of the name. Instead of total disagreement weight, they get assigned a small positive agreement weight. Incidentally, for the first name pairs, the largest weight differences involved short strings, especially initials. When comparing, for example, "L C" and "LC", the J-W comparator results in a low disagreement weight while the edit and adaptive comparators produce a high agreement weight.

Edit Weight > J-W Weight

Last Name Match Pairs		Scores		Weights		Diff
Name 1	Name 2	Edit	J-W	Edit	J-W	
COHEE-WHEELER	WHEELER	0.7692	0.6264	1.3701	−2.9108	4.2809
SHELL GLADNEY	GLADNEY	0.7692	0.3352	1.3701	−2.9108	4.2809
STRUB-TURNAGE	TURNAGE	0.7692	0.4860	1.3701	−2.9108	4.2809
JONES-ATKINS	ATKINS	0.7500	0.5500	1.0825	−2.9108	3.9933
FOSTER-ARNITZ	ARNITZ	0.7308	0.4021	0.7948	−2.9108	3.7056
JONES GOSBY	GOSBY	0.7273	0.5855	0.7425	−2.9108	3.6533
JONES	PARKER JONES	0.7083	0.4561	0.4592	−2.9108	3.3700
SKAGGE SLOAN	SLOAN	0.7083	0.5506	0.4592	−2.9108	3.3700
COAD	JACOAD	0.8333	0.8056	2.3290	−0.5110	2.8401

(12)

In Table (13) we see the last name non-match pairs for which the edit weight most exceeds the J-W weight. While these pairs may not be matches, the edit comparator is registering a recognizable similarity between the strings.

Edit Weight > J-W Weight

Last Name Non-Match Pairs		Scores		Weights		
Name 1	Name 2	Edit	J-W	Edit	J-W	Diff
ORE	MOORE	0.8000	0.5644	1.8304	−2.9108	4.7412
WARD	EDWARD	0.8333	0.7222	2.3290	−2.4065	4.7356
EDWARDS	WARD	0.7857	0.6905	1.6167	−2.9108	4.5275
HERMANN	MANN	0.7857	0.5821	1.6167	−2.9108	4.5275
WOODARD	WARD	0.7857	0.6345	1.6167	−2.9108	4.5275
WARD	STEWARD	0.7857	0.0000	1.6167	−2.9108	4.5275
BAUER	WOLFBAUER	0.7778	0.0000	1.4980	−2.9108	4.4088
BELL	CAMPBELL	0.7500	0.4958	1.0825	−2.9108	3.9933
WARD	POWLLARD	0.7500	0.5333	1.0825	−2.9108	3.9933
MAY	MCNALLY	0.7143	0.6508	0.5482	−2.9108	3.4590

(13)

In Table (14) we see the last name match pairs where the J-W weight most exceeds the edit weight. The differences here are not as large as in Table(12), and although these pair come from match records, only about half of them look like likely matching strings, so that the justification for the higher weight is not obvious..

J-W Weight > Edit Weight

Last Name Match Pairs		Scores		Weights		
Name 1	Name 2	Edit	J-W	Edit	J-W	Diff
JONES	JOHNSON	0.6476	0.8738	−0.4490	1.0419	−1.4909
THOMPSON	JOHNSON	0.5357	0.8011	−2.1230	−0.6127	−1.5102
HAINES	HAYMES	0.6667	0.8880	−0.1641	1.3642	−1.5283
DOLIE	DOYLE	0.7333	0.9347	0.8332	2.4257	−1.5925
GANES	GAINS	0.7333	0.9347	0.8332	2.4257	−1.5925
JONES	JAMES	0.6000	0.8488	−1.1613	0.4726	−1.6339
WRATHY	WORTHY	0.7500	0.9475	1.0825	2.7176	−1.6351
CURRUTHERS	CORORRETHRS	0.6227	0.8697	−0.8214	0.9474	−1.7688
COVINGTON	COUTION	0.5794	0.8463	−1.4700	0.4168	−1.8868
CHEATHER	CHEETMA	0.5893	0.8932	−1.3216	1.4821	−2.8037

(14)

In Table (15) we have the largest differences with the J-W weight exceeding the edit weight for non-match last name pairs. Here most of the differences are larger than in Table (14), generally raising a near total disagreement edit weight to a near zero to slightly positive J-W weight. The name pairs generally look convincingly unmatched.

J-W Weight > Edit Weight

Last Name Non-Match Pairs		Scores		Weights		
Name 1	Name 2	Edit	J-W	Edit	J-W	Diff
ADAMS	KAMADULSKI	0.4833	0.8216	−2.9065	−0.1468	−2.7597
CARNAGHI	CUNNINGHAM	0.4500	0.8252	−2.9108	−0.0644	−2.8464
OWENS	JONES	0.5000	0.8375	−2.6572	0.2155	−2.8727
LEWIS	WILLIS	0.4667	0.8274	−2.9108	−0.0153	−2.8955
WILSON	LEWIS	0.3833	0.8274	−2.9108	−0.0153	−2.8955
HOLLINS	ALLISON	0.4464	0.8286	−2.9108	0.0124	−2.9232
STEWART	ATWATER	0.4464	0.8286	−2.9108	0.0124	−2.9232
THOMPSON	MCPHERSON	0.4792	0.8332	−2.9108	0.1174	−3.0282
BEANS	BASKIN	0.4167	0.8347	−2.9108	0.1525	−3.0633
WALKER	FOWLER	0.5000	0.8516	−2.6572	0.5360	−3.1932
FOWLER	WALTER	0.5000	0.8516	−2.6572	0.5360	−3.1932
STOKER	KUSTER	0.5000	0.8516	−2.6572	0.5360	−3.1932
SPINKS	PERKINS	0.4762	0.8405	−2.9108	0.2844	−3.1952
HAMILTON	HOLMAN	0.4375	0.8458	−2.9108	0.4044	−3.3152
NICHOLSON	JOHNSON	0.5079	0.8636	−2.5385	0.8088	−3.3472

(15)