RESEARCH REPORT SERIES
(*Statistics #2005-05*)

Evaluating String Comparator Performance for Record Linkage

William E. Yancey

Statistical Research Division
U.S. Census Bureau
Washington, DC 20233

Report Issued: June 13, 2005

Evaluating String Comparator Performance for Record Linkage

William E. Yancey
Statistical Research Division
U.S. Census Bureau

June 9, 2005

Abstract

We compare variations of string comparators based on the Jaro-Winkler comparator and edit distance comparator. We apply the comparators to Census data to see which are better classifiers for matches and non-matches, first by comparing their classification abilities using a ROC curve based analysis, then by considering a direct comparison between two candidate comparators in record linkage results.

1 Introduction

We wish to evaluate the performance of some string comparators and variations for use in record linkage software for Census Bureau data. For record linkage, under the conditional independence assumption, we compute a comparison weight for two records from the sum of the comparison weights of the individual matching fields. If we designate that the values of a matching field agree for two records by $\gamma = 1$ and that they disagree by $\gamma = 0$, then we define the agreement weight for the two fields by

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

and the disagreement weight by

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

where the probabilities are conditioned by whether the two records do in fact belong to the set M of true matches or the set U of true non-matches. If we wish to use a string comparator for the matching field with alphabet Σ, we generally use a *similarity function*

$$\gamma : \Sigma^* \times \Sigma^* \to [0, 1]$$

where $\gamma(\alpha, \beta) = 1$ when the strings α, β are identical. We then use an interpolation function w,

$$w : [0, 1] \rightarrow [d_w, a_w]$$

to assign a comparison weight $w(x)$ to a pair of strings α, β where the interpolation function is increasing with $w(1) = a_w$.

We next describe the string comparator functions that we used for this study. We then discuss the data sets that were used to test the comparators. Then we discuss how we interpreted the results of this data to try to evaluate the classification power of each of the string comparators. We also look at the difference that the string comparator choice can make in a matching situation.

2 The String Comparator Functions

In the following, let α, β be strings of lengths m, n respectively with $m \leq n$.

2.1 The Jaro-Winkler String Comparators

2.1.1 The Basic Jaro-Winkler String Comparator

The Jaro-Winkler string comparator [3] counts the number c of common characters between two strings and the number of transpositions of these common characters. A character a_i of string α and b_j of string β are considered to be common characters of α, β if $a_i = b_j$ and

$$|i - j| < \left\lfloor \frac{n}{2} \right\rfloor,$$

the greatest integer of half the length of the longer string. A character of one string is considered to be common to at most one character in the other string. The number of transpositions t is determined by the number of pairs of common characters that are out of order. The number of transpositions is computed as the greatest integer of half of the number of out-of-order common character pairs. The Jaro-Winkler similarity value for the two strings is then given by

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

unless the number of common characters $c = 0$, in which case the similarity value is 0.

Example 1 *Consider the strings (b,a,r,n,e,s) and (a,n,d,e,r,s,o,n). The search range distance d*

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

for common characters is $d = 3$*, since the longer string length is* 8*. The set of* 5 *common characters is thus {a,r,n,e,s}, which occur in the second string in the*

order (a,n,e,r,s), so the middle 3 *characters are out of position, which counts as* 1 *transposition. Thus the basic J-W score is given by*

$$x = \frac{1}{3}\left(\frac{5}{6} + \frac{5}{8} + \frac{4}{5}\right) = \frac{271}{360} \doteq 0.75280.$$

There are three modifications to this basic string comparator that are currently in use.

2.1.2 Similar Characters

The string comparator program contains a list of 36 pairs of characters that have been judged to be similar, so that they are more likely to be substituted for each other in misspelled words. After the common characters have been identified, the remaining characters of the strings are searched for similar pairs (within the search distance d). Each pair of similar characters increases the count of common characters by 0.3. That is the similar character count is given by

$$c_s = c + 0.3s$$

where s is the number of similar pairs. The basic Jaro-Winkler formula is then adjusted by

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

For instance the strings *abc* and *ebc* have 2 common characters and the similar pair (a, e), so that the new score is given by

$$\begin{aligned} x_s &= \frac{1}{3}\left(\frac{2}{3} + \frac{2}{3} + 1\right) + \frac{1}{3}\left(\frac{0.3}{3} + \frac{0.3}{3}\right) \\ &= \frac{7}{9} + \frac{1}{15} \\ &= \frac{38}{45} \end{aligned}$$

The adjusted score is $\frac{38}{45} \doteq 0.8444$ instead of the basic score $\frac{7}{9} \doteq 0.778$. In our previous example the only unmatched character of the first string is b and the only candidate unmatched character in the second string is d, and (b, d) is not included in the similar character list, so no adjustment to the basic score is made.

2.1.3 Common Prefix

This adjustment increases the score when the two strings have a common prefix. If p is the length of the common prefix, up to 4 characters, then the score x is adjusted to x_p by

$$x_p = x + \frac{p\,(1-x)}{10}.$$

2.1.4 Longer String Adjustment

Finally there is one more adjustment in the default string comparator that adjusts for agreement between longer strings that have several common characters besides the above agreeing prefix characters. The conditions for using the adjustment are

$$\begin{aligned} m &\geq 5 \\ c-p &\geq 2 \\ c-p &\geq \frac{m-p}{2} \end{aligned}$$

That is,

1. Both strings are at least 5 characters long;
2. There are at least two common characters besides the agreeing prefix characters;
3. We want the strings outside the common prefixes to be fairly rich in common characters, so that the remaining common characters are at least half of the remaining common characters of the shorter string.

If all of these conditions are met, then length adjusted weight x_l is computed by

$$x_l = x + (1-x)\frac{c-(p+1)}{m+n-2(p-1)}$$

In our example, the two names have no common prefix, but they satisfy the conditions for the long string adjustment, so the adjusted score x_l is given by

$$\begin{aligned} x_l &= \frac{271}{360} + \left(1 - \frac{271}{360}\right)\frac{5-1}{6+8+2} \\ &= \frac{391}{480} \\ &\doteq 0.8146 \end{aligned}$$

2.2 Edit Distance String Comparators

We wished to compare the Jaro-Winkler string comparators with some string comparators based on edit distance. All of the edit distance type comparator values are computed using a dynamic programming algorithm that computes the comparison value in $O(mn)$ time.

2.2.1 Standard Edit Distance

The standard edit distance (or Levenshtein distance) [1] between two strings is the minimum number of edit steps required to convert one string to the other, where the allowable edit steps are insertion, deletion, and substitution. If we

let α_i be the prefix of α of length i, β_j be the prefix of β of length j, and ε be the empty string, then we can initialize the edit distance algorithm with the distances

$$\begin{aligned} e\left(\alpha_i, \varepsilon\right) &= i \\ e\left(\varepsilon, \beta_j\right) &= j \\ e\left(\varepsilon, \varepsilon\right) &= 0 \end{aligned}$$

indicating the number of insertions/deletions to convert a string to the empty string. We can then build up the cost of converting longer prefixes by computing

$$e\left(\alpha_i, \beta_j\right) = \min \begin{cases} e\left(\alpha_{i-1}, \beta_j\right) + 1 \\ e\left(\alpha_i, \beta_{j-1}\right) + 1 \\ \begin{cases} e\left(\alpha_{i-1}, \beta_{j-1}\right) & \text{if } a_i = b_j \\ e\left(\alpha_{i-1}, \beta_{j-1}\right) + 1 & \text{if } a_i \neq b_j \end{cases} \end{cases} \tag{1}$$

where a_i denotes the i^{th} character of α and b_j is the j^{th} character of β. The final minimum edit cost is then given by $e\left(\alpha, \beta\right) = e\left(\alpha_m, \beta_n\right)$.

While the edit distance function is a true metric on the space of strings, it is not a similarity function. We note that the maximum edit length between two strings is n (m substitutions and $n - m$ insertions/deletions), so that the comparator score

$$x_e = 1 - \frac{e}{n}$$

defines a similarity function for string pairs α, β where e is the edit distance between the strings. One can check that in our example, a minimal edit path has 6 edits, such as

$$\left(b, \varepsilon\right) a \left(r, n\right) \left(n, d\right) e \left(\varepsilon, r\right) s \left(\varepsilon, o\right) \left(\varepsilon, n\right)$$

resulting in an edit similarity score of

$$x_e = 1 - \frac{6}{8} = \frac{1}{4}.$$

We not that character order is important to edit distance, so that the three common characters that are out of order result in three edits.

2.2.2 Longest Common Subsequence

The length of the longest common subsequence (lcs) of two strings can also be computed by the same algorithm [1], except that this time the only possible edit steps are insertion and deletion, so that the recursive function is

$$e\left(\alpha_i, \beta_j\right) = \min \begin{cases} e\left(\alpha_{i-1}, \beta_j\right) + 1 \\ e\left(\alpha_i, \beta_{j-1}\right) + 1 \\ \quad e\left(\alpha_{i-1}, \beta_{j-1}\right) \quad \text{if } a_i = b_j \end{cases} \tag{2}$$

Clearly the longest possible common subsequence length for our strings is m, so we can define a longest common subsequence similarity function by

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

where l is the length of the longest common subsequence of the two strings. In our example, the maximum lcs (a, n, e, s) length is 4, so the similarity score is

$$x_c = \frac{4}{6} = \frac{2}{3}.$$

2.2.3 Coherence Edit Distance

As we have seen, both edit distance and lcs length depend strictly on the order of the characters in the strings. This is because the defining recursion functions (1,2) that determine the cost of the current prefix pair depend only on the last characters of the two prefixes, *i.e.* whether they are equal or not. There is no checking to see whether one of these characters occurred somewhere earlier in the other prefix. The Jaro-Winkler string comparator allows common characters to be out of order with a penalty for transpositions. If the edit distance recursion looked back over earlier characters in the prefixes, then a contextual edit score could be given. In [2], Jie Wei uses Markov field theory to develop such a recursion function, which looks like

$$C\left(\alpha_i, \beta_j\right) = \min \begin{cases} C\left(\alpha_{i-1}, \beta_j\right) + 1 \\ C\left(\alpha_i, \beta_{j-1}\right) + 1 \\ \min_{\substack{1 \le a \le \mathsf{N} \\ 1 \le b \le \mathsf{N}}} \left\{ C\left(\alpha_{i-a}, \beta_{j-b}\right) + V_{a,b}\left(\alpha_i, \beta_j\right) \right\} \end{cases}$$

where $V_{a,b}\left(\alpha_i, \beta_j\right)$ is an edit cost potential function and N is a number that indicates the degree of coherence of the strings, *i.e.* the amount of context that should be considered when computing the edit cost. For $V_{a,b}\left(\alpha_i, \beta_j\right)$, we consider the substrings $\alpha_{i-a,i}$ and $\beta_{j-b,j}$ and let c be the number of common characters to these two substrings. Denoting $t = a + b$, we can express Jie Wei's coherence edit potential function as

$$V_{a,b}\left(\alpha_i, \beta_j\right) = \begin{cases} \frac{3}{4}\left(t - \frac{2}{3}\right) - c & \text{if } t \text{ is even} \\ \frac{3}{4}\left(t - 1\right) - c & \text{if } t \text{ is odd} \end{cases}$$

He also chooses $\mathsf{N} = 4$ as a reasonable coherence index for words. With this choice of N we can display all possible values of (a, b) and the corresponding

range of V_{ab}

(a,b)	t	$\max V_{ab}$	$\max c$	$\min V_{ab}$
$(1,1)$	2	1	1	0
$(1,2)$	3	1.5	1	0.5
$(1,3)$	4	2.5	1	1.5
$(1,4)$	5	3	1	2
$(2,2)$	4	2.5	2	0.5
$(2,3)$	5	3	2	1
$(2,4)$	6	4	2	2
$(3,3)$	6	4	3	1
$(3,4)$	7	4.5	3	1.5
$(4,4)$	8	5.5	4	1.5

Returning to our example, the coherence edit distance between *barnes* and *anderson* is 3.5. The edit sequence with the costs is

$\binom{ba}{a}$	$\binom{rne}{nder}$	$\binom{s}{so}$	$\binom{\varepsilon}{n}$
0.5	1.5	0.5	1

The coherence edit distance is always less than or equal to the standard edit distance, so the maximum possible distance is still the length of the longer string, and we can define a coherence edit similarity score by

$$x_w = 1 - \frac{C}{n}$$

where C is the coherence edit distance of the two strings. In the above example, we have

$$x_w = \frac{4.5}{8} = 0.5625.$$

2.2.4 Combination Edit Distance

When we began studying and evaluating string comparators using the approach discussed in Section 5, we found that the edit distance similarity function did not perform as well as the J-W comparator. This may be because it does not use enough information from the strings. In particular, it does not consider the length of the shorter string. Thus we tried combining the edit distance and the *lcs* comparators by averaging them

$$x_{ec} = \frac{1}{2}\left(\left(1 - \frac{e}{n}\right) + \frac{l}{m}\right)$$

which seemed to produce better results. The example string pair has a combined edit score of $\frac{11}{24} \doteq 0.4583$. There could be a more optimal weighting of the two comparators.

2.2.5 Combination Coherence Edit Distance

We also considered the effect of combining the coherence edit distance and the *lcs* comparators.

$$x_{wc} = \frac{1}{2}\left(\left(1 - \frac{C}{n}\right) + \frac{l}{m}\right).$$

The example produces a combined score of $\frac{59}{96} \doteq 0.6146$.

2.3 Hybrid Comparators

Our initial analysis of the results of our experiment led us to consider combining a J-W comparator with an edit distance type comparator. We will consider this development after we describe the experiment.

3 Data Sets

There does not appear to be any theoretical way to determine which is the best string comparator. In fact, there does not appear to be a clear meaning of "best" other than the comparator that performs best on a given application. We therefore want to conduct an experiment to see which string comparator does the best job for the application of record linkage of Census data. Since the string comparator is just one component of the record linkage procedure, we first try to isolate the string comparator's contribution.

At the Census Bureau, we have some test decks that are pair of files where the matches have been clerically determined. One large pair of files come from the 2000 Census and the ACE follow-up. These files each have $606,411$ records of persons around the country where each record of one file has been matched with one record of the other. There are also three smaller pairs of files from the 1990 Census and the PES follow-up. Each of these files is of persons in the same geographic area where not all of the records have matches.

The data sets were formed by bringing together the records that were identified as matches and writing out the pairs of last names or first names whenever the two name strings were not identical. We then removed all duplicate name pairs from the list. From the 2000 data we obtained a file of $65,325$ distinct first name pairs and $75,574$ distinct last name pairs. From the three 1990 files combined we obtained three files of 942, 1176, and 2785 distinct first and last name pairs.

4 The Computation

The purpose of the string comparator in record linkage is to help us distinguish between pairs of strings that probably both represent the same name and pairs of strings that do not. For the sets M of matched pairs, we will use data sets of nonidentical name pairs from matched records. Our data sets do not necessarily contain only string pairs representing the same name, since some records may

have been linked based on information of other fields than this name field. However, they should tend to have similarities that one may subjectively judge to suggest that they refer to the same name. For our sets U of unmatched pairs, we will take the set of cross pairs of every first member name in the set paired with every second member name *other* than its match. We may think of these as unmatched pairs, but they are really more like random pairs, since there can be pairs of names in U which match exactly. Thus we can never completely separate the set M from the set U, but the test will be which string comparator can include the most elements of M with the fewest elements of U.

For each string comparator under examination, we compute the string comparator value of each first member name with each second member name. The sets of Census 2000 names are too large to store the resulting comparator values, so we split each of the sets into 24 subsets of name pairs, The first name pair subsets have 2722 pairs (2719 for the last one) and the last name pairs have 3149 pairs (3147 for the last one). Thus we compute the values of 13 string comparators, 8 J-W comparators with all combinations of the three modifications and 5 edit distance type comparators, of all cross pairs of strings in 51 sets of name pairs to generate our string comparator output data.

5 Analysis of Scores

We now face the problem of determining how to use this data to measure the performance of the string comparators. We can view the problem as a binary classification problem: a string pair either belongs to M or it belongs to U. One tool for analyzing classification effectiveness is the ROC curve. The ROC (receiver operating characteristic) curve originated for use in signal detection, but is now commonly used in medicine to measure the diagnostic power of a test. A list of references can be found on [4]. If we let γ be a string comparator similarity function, we measure the discriminatory power of the string comparator with the parameterized curve

$$\left(\Pr\left(\gamma \geq t|U\right), \Pr\left(\gamma \geq t|M\right)\right), \quad t \in [0,1]$$

in the unit square. The resulting ROC curve is then independent of the parameterization. This is sometimes referred to as plotting sensitivity against 1− specificity. If we denote the probability density of the M condition by $p_M(t)$ and the density conditioned on U by $p_U(t)$ so that

$$\begin{aligned} \frac{d}{dt}\Pr\left(\gamma \geq t|M\right) &= -p_M(t) \\ \frac{d}{dt}\Pr\left(\gamma \geq t|U\right) &= -p_U(t) \end{aligned}$$

then we see that the slope of the tangent to the ROC curve is

$$\frac{dy}{dx} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{p_M(t)}{p_U(t)}$$

the likelihood ratio of the two distributions. The diagnostic tool that is used is the AUC, the area under the ROC curve. To interpret this AUC, we define the random variables

$$\begin{aligned} X &: M \to [0,1] \\ Y &: U \to [0,1] \end{aligned}$$

to be the string comparator value for a pair drawn from M or U respectively, which have probability density functions p_M and p_U respectively, then

$$AUC = \int_0^1 \Pr(\gamma \geq t|M)\ d\Pr(\gamma \geq t|U) = \Pr(X \geq Y),$$

the probability that a randomly chosen element of M will have a higher score than a randomly chosen element of U. Thus an $AUC = 1$ would indicate that the string comparator γ is a perfect discriminator between M and U, and an $AUC = \frac{1}{2}$ would indicate that γ has no discriminating power whatsoever between M and U. Hence the nearer AUC is to 1, the more effective the discriminator between the two sets.

We used our data to compute the AUC for each of our string comparators, but then we realized that for our application, the full AUC is not a very relevant statistic. In our record linkage program, the string comparator similarity score is linearly interpolated to produce an agreement weight between the full agreement weight and the full disagreement weight. When the interpolation value is less than the disagreement weight, the disagreement weight is assigned. Thus all similarity scores below a cutoff value are treated the same, as indicating that the sting pairs are in U. The only discrimination happens for string comparators above this cutoff. Thus for sufficiently small values of $\alpha \in [0,1]$, we instead looked at values of

$$\frac{1}{\Pr(\gamma \geq t_\alpha|U)} \int_0^\alpha \Pr(\gamma \geq t|M)\ d\Pr(\gamma \geq t|U)$$

where $t_a \in [0,1]$ such that $\Pr(\gamma \geq t_\alpha|U) = \alpha$. Since

$$\begin{aligned} \int_0^\alpha \Pr(\gamma > t|M)\ d\Pr(\gamma \geq t|U) &= \int_{t_\alpha}^1 \Pr(\gamma \geq t|M)\, p_U(t)\, dt \\ &= \int_{t_\alpha}^1 \int_t^1 p_M(s)\, p_U(t)\, ds dt \end{aligned}$$

we see that this is the probability that $X \geq Y$ and $Y \geq t_\alpha$. Thus

$$\frac{1}{\Pr(\gamma \geq t_\alpha|U)} \int_0^\alpha \Pr(\gamma \geq t|M)\ d\Pr(\gamma \geq t|U) = \Pr(X \geq Y|Y \geq t_\alpha).$$

As $\alpha \nearrow 1$, then $t_a \searrow 0$, and we see that the interpretation agrees with the standard one in the limit.

The weight interpolation function currently in use in the matching software is

$$w = a_w - 4.5\,(a_w - d_w)\,(1 - s)$$

where s is the string comparator score from the Jaro-Winkler comparator using all three modifications. With this interpolation, we see that we will get the full disagreement weight $w = d_w$ for

$$s_- = \frac{7}{9} \doteq 0.778.$$

When we look at the J-W comparator scores as a function of the "error rate" $\Pr(\gamma \geq t|U) = \alpha$, we see that we are past this boundary score by the time $\alpha = 0.02$, sometimes by $\alpha = 0.01$. Thus we are interested in only a small sliver of the total area under the ROC curve. Moreover, the region corresponding to a positive agreement weight is smaller still. For example, if we have parameters that result in $d_w = -a_w$ (*i.e.* $\Pr(\gamma = 1|M) + \Pr(\gamma = 1|U) = 1$), then the agreement weight $w = 0$ when

$$s_+ = \frac{8}{9} \doteq 0.889.$$

Suppose that we designate by p_+, p_- the "error rate" probabilities for the full Jaro-Winkler (all options) scores where for which $\Pr(\gamma > s_+|U) = p_+$ and $\Pr(\gamma > s_-|U) = p_-$. These boundary error probabilities remain mostly consistent within the related groups of data sets: the three sets of names from 1990, the 24 sets of first names from 2000, and all but 3 of the sets of last names from 2000. The last three sets of last names are consistent with each other but have a different error/score profile than the first 21 sets. This appears to be because these last sets consist mostly of Hispanic last names and the random cross pairs contain a higher proportion of incidental exact matches. Approximate values of these cutoff error rates are given in the following table.

Data Sets	p_+	p_-
1990 Names	0.0014	0.017
2000 First Names	0.0012	0.014
Main Group 2000 Last Names	0.0004	0.01
Subgroup 2000 Last Names	0.006	0.022

In all cases we see that for the given weighting function for the full J-W string comparator results in a very small part of the range $0 \leq \Pr(\gamma > t|U) \leq 1$ is relevant for the diagnostic power of the ROC curve. We will consider only such restrictive ranges when comparing the relative strengths of the candidate string comparators. That is, for restrictive values of x_i, for the corresponding value of t_i, where

$$\Pr(\gamma > t_i|U) = x_i$$

we consider the value of

$$\frac{1}{\Pr(\gamma > t_i|U)} \int_0^{x_i} \Pr(\gamma \geq t|M)\; d\Pr(\gamma \geq t|U)$$

	2021	3031	STL
JW000	0.417	0.420	0.458
JW001	0.408	0.407	0.442
JW010	0.433	0.438	0.481
JW100	0.420	0.431	0.470
JW011	0.425	0.426	0.462
JW101	0.428	0.436	0.472
JW110	0.434	0.447	0.487
JW111	0.440	0.452	0.489

Table 1: Sensitivity at Error Rate 0.0012

5.1 The Jaro-Winkler String Comparators

The basic Jaro-Winkler string comparator has three optional adjustments: common prefix, similar characters, and long string suffix. We wish to see what effect these adjustments has on the performance of the string comparator. We denote the J-W variations by JW*xyz*, where *x,y,z* are Boolean values for the use of respectively the

- prefix,
- similar character,
- long suffix adjustments

5.1.1 The 1990 Names

The three 1990 name files give similar results. Basically there is not much difference between the string comparator variations. The weakest versions are JW001 and JW011 which have the long string suffix adjustment without the common prefix adjustment. Since the long string suffix adjustment is designed to supplement the common prefix adjustment, these empirical results agree with intuition. To illustrate, we can look at the graph of the average sensitivity with the specificity for comparator values that should be in approximately the range of positive agreement weight. We show the graph for the names from the data set 3031 in Fig. 1. The other two sets are similar. At the high score end the average sensitivity values are very close. In Table 1 we show numerical values for specificity $\Pr(\gamma > t|U) = 0.0012$, where there is some separation toward the low end of the positive matching weight range. Except in the two cases JW001 and JW011 noted above, we can see that each of the three adjustments increases the sensitivity with the smallest contribution. Thus the highest sensitivity is achieved by JW111, although JW110 is very close. In Fig. 2 we show the entire range of selectivity values over which anything more than a total disagreement weight should result. The sensitivities are close with the usual two comparators at the bottom. The sensitivity values for $\Pr(\gamma > t|U) = 0.01$ around the middle of the negative agreement weight range are given in Table 2. In this region, the

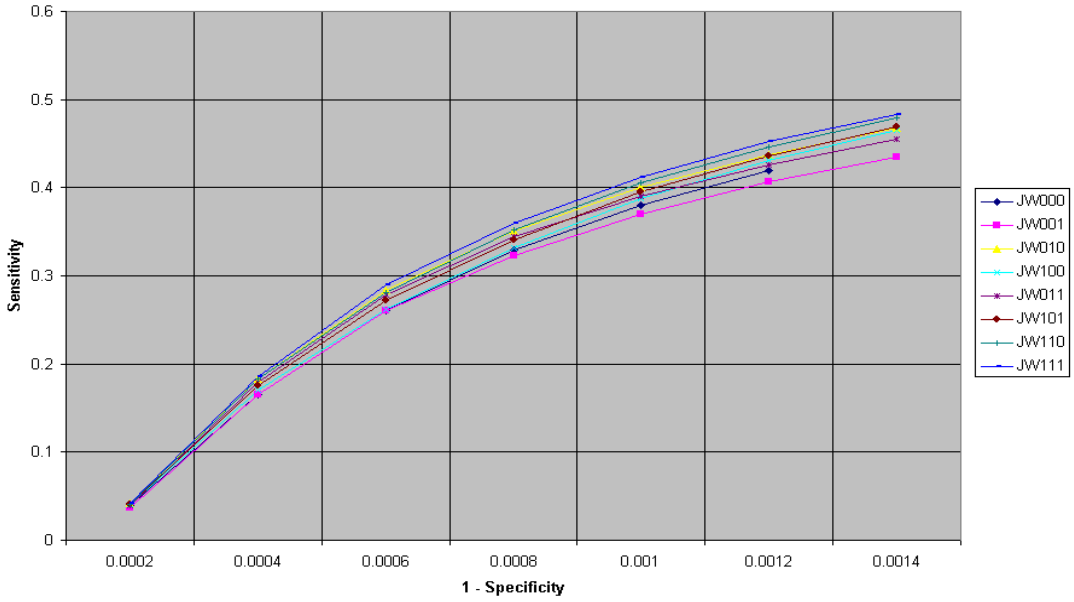

Figure 1: Positive Weight Range for J-W Scores from 1990 3031 File

	2021	**3031**	**STL**
JW000	0.707	0.709	0.732
JW001	0.687	0.682	0.707
JW010	0.714	0.718	0.741
JW100	0.715	0.730	0.749
JW011	0.695	0.690	0.716
JW101	0.711	0.719	0.739
JW110	0.722	0.736	0.754
JW111	0.718	0.728	0.746

Table 2: Average Sensitivity at Error Rate 0.01

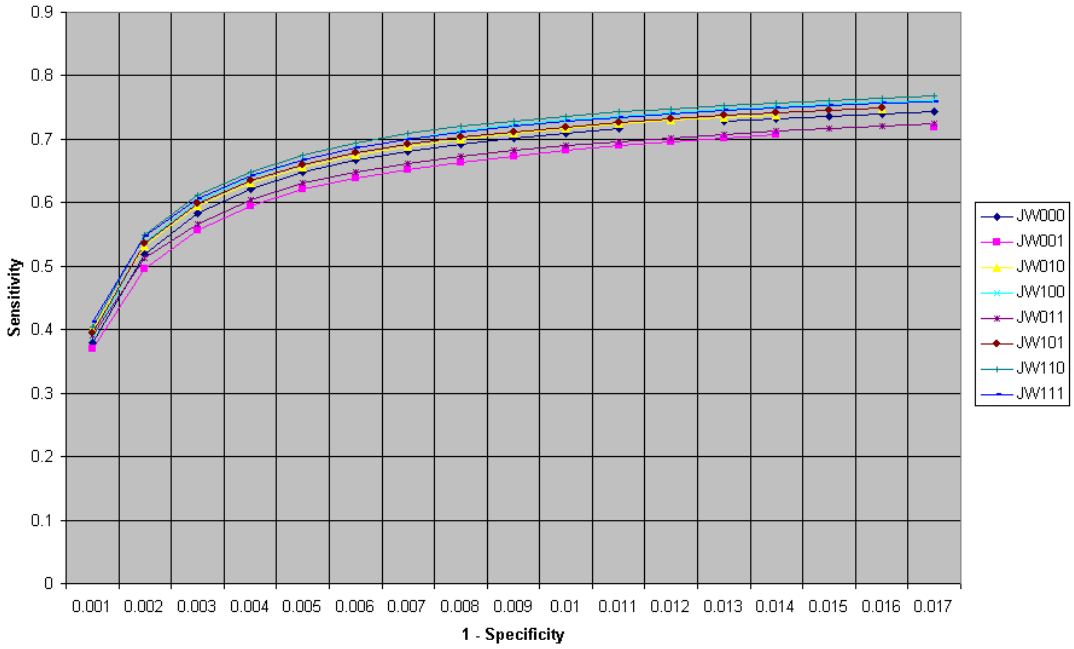

Figure 2: Full Weight Range for J-W Scores from 1990 3031 File

prefix adjustment increases the sensitivity, the similar character adjustment increases the sensitivity slightly, and the long suffix adjustment decreases the sensitivity slightly. Thus the JW100 comparator has the highest sensitivity, with the JW111 sensitivity just slightly below.

5.1.2 The 2000 First Names

Looking at the 24 sets of first name pairs from 2000, we again see that there is not too much difference among the J-W variations. There is strong consistency between the JW111 score t and the specificity $\Pr(\gamma > t|U)$ across all of the 24 sets, so we present the average sensitivities across the 24 sets. The mainly positive agreement weight range is shown in Fig. 3. The total weight range is shown in Fig. We can compare sensitivity values for specificity values in the positive and negative weight ranges in Table 3. We see that the prefix adjustment increases sensitivities, the similar character adjustment slightly increases sensitivities, and the suffix adjustment slightly decreases sensitivities. Thus the comparator JW110 has the highest sensitivity, but all four comparators with the prefix adjustment are barely distinguishable.

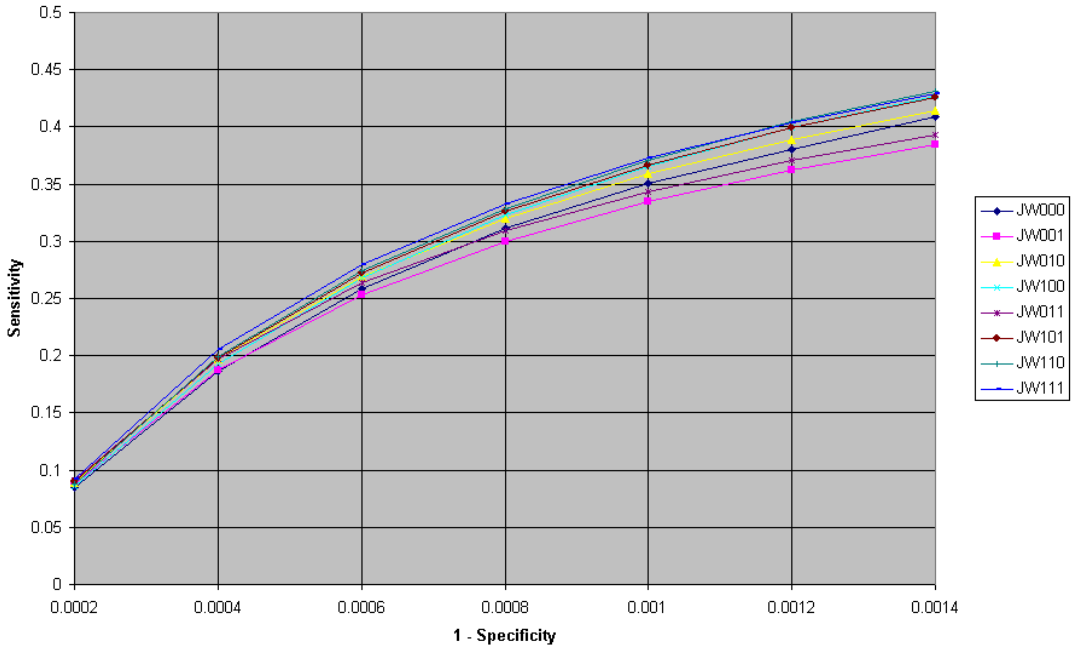

Figure 3: Positive Weight Range for Average J-W Scores for 2000 First Names

	1− Specificity	
	0.0012	0.01
JW000	0.381	0.646
JW001	0.363	0.613
JW010	0.389	0.647
JW100	0.400	0.659
JW011	0.371	0.613
JW101	0.399	0.651
JW110	0.404	0.663
JW111	0.404	0.656

Table 3: Sensitivities for 2000 First Name Pairs

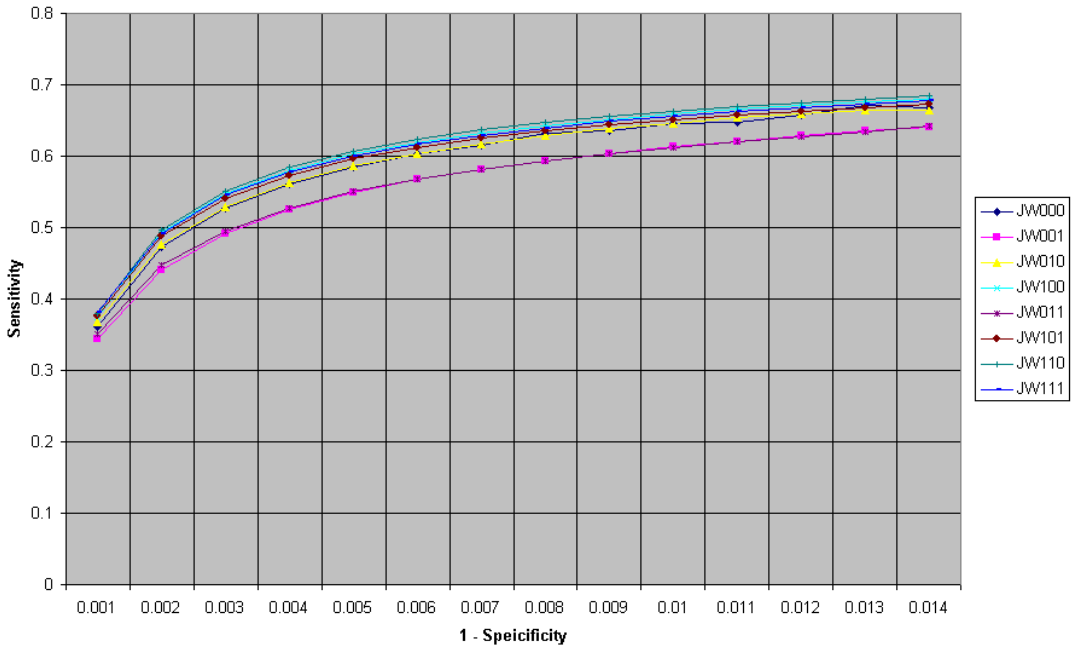

Figure 4: Total Weight Range for J-W String Comparators for 2000 First Names

	1− Specificity	
	0.0002	0.005
JW000	0.266	0.699
JW001	0.266	0.682
JW010	0.275	0.700
JW100	0.274	0.705
JW011	0.274	0.681
JW101	0.280	0.704
JW110	0.280	0.706
JW111	0.286	0.705

Table 4: First Name Averages for J-W Comparators, 21 Sets

5.1.3 The 2000 Last Names

As mentioned earlier, the last three last name files have different ROC curve results that the first 21 last name files. For the first 21 files, the specificity bounds corresponding to positive and negative weight scores are quite low. For these 21 sets, the positive weight range ends at about $\Pr(\gamma > t|U) = 0.0002$ and the whole weight adjustment range ends by $\Pr(\gamma > t|U) = 0.01$. In Table 4 below we give the sensitivity values for the end of the positive weight range and the middle of the total weight range. In the positive range, all three adjustments produce some sensitivity increase, so the highest sensitivity is obtained by using all three adjustments. In the middle of the negative weight range, the similar character adjustment produces a very slight increase and the suffix adjustment produces a very slight decrease, so the highest sensitivity is obtained by using the prefix and similar character adjustments. However, except for the usual two cases that use the suffix adjustment without the prefix adjustment, the sensitivities are all nearly the same, as can be seen in Fig.5

The last three sets of last name pairs have a very different specificity/sensitivity profile. The positive weight range ends about at $\Pr(\gamma > t|U) = 0.006$ and the negative weight scaling lasts until $\Pr(\gamma > t|U) = 0.017$, considerably higher values than for the first 21 sets. We can see sample values in the middle of the positive and negative weight zones in Table 5 and the range of sensitivities is shown in Fig. 6. In this case we see that the prefix adjustment results in some improvement, but the the similar character and suffix adjustments result in lower sensitivities. Furthermore, there is a greater difference between the sensitivity values than there was for the other sets. The highest sensitivity belongs to the J-W comparator using the prefix adjustment, but the one using no adjustments is not far behind. The other comparator with comparable values uses the prefix and the similar character adjustments.

The comparator results for the first 21 sets of pairs of last names and for the last 3 sets differ in a few ways. The same JW111 score results in higher values of both $\Pr(\gamma > t|M)$ and $\Pr(\gamma > t|U)$. In the first 21 sets, the three adjustments all seem to help, but they do not produce much difference. In the last three sets, the only the prefix adjustment helps while the other two adjustments hinder,

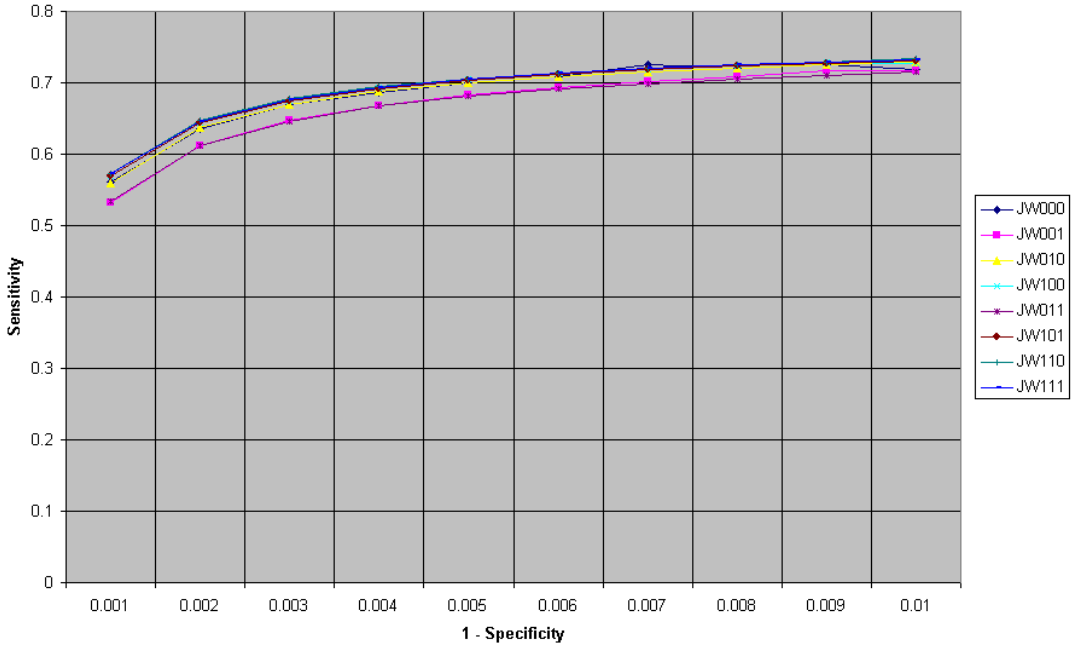

Figure 5: Last Name Sensitivity Averages for J-W Comparators for 21 Sets

	1− Specificity	
	0.002	0.01
JW000	0.520	0.806
JW001	0.394	0.696
JW010	0.457	0.778
JW100	0.557	0.817
JW011	0.357	0.643
JW101	0.443	0.775
JW110	0.494	0.796
JW111	0.399	0.750

Table 5: Last Names Final 3 Sets, J-W Averages

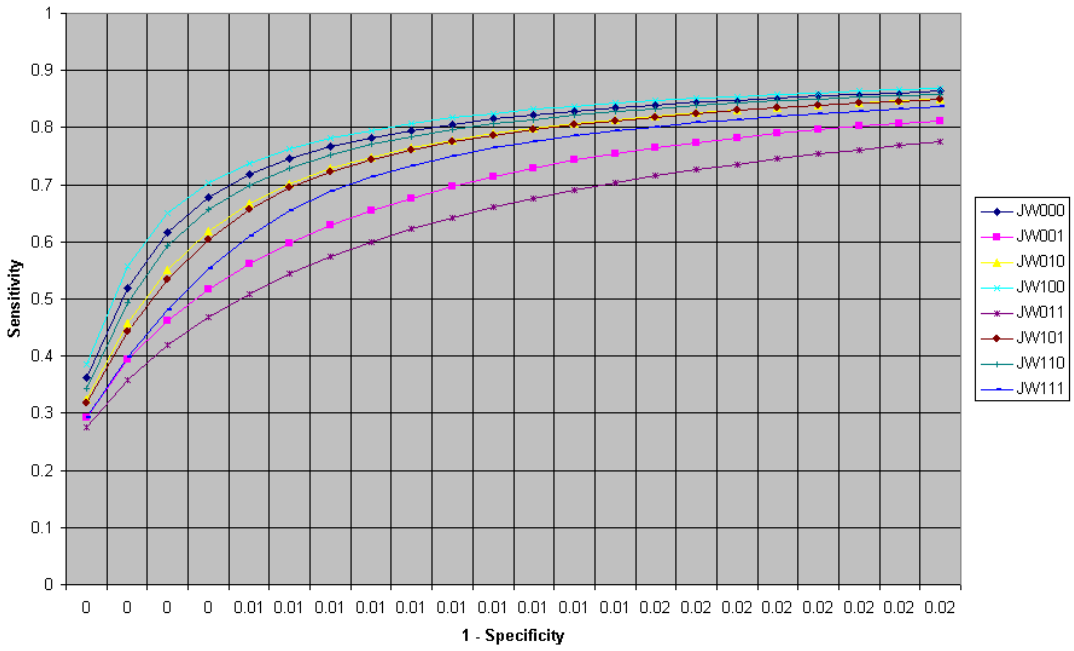

Figure 6: Averages of J-W Sensitivities for the Last Three Sets of Last Names

and there are large differences between the comparator sensitivities. The reason for this appears to be that the last 3 sets consist of predominantly Hispanic last names. A large source of error is that a lot of surnames are double names (e.g. *Garcia Marquez*) which might appear in full in one file and only one of the names (*Garcia* or *Marquez*) might appear in the other file. Another difference in the statistics can be affected by the more limited number of distinct surnames, so that when we form the set of all non-matched cross pairs, a higher proportion of these are duplicate pairs. Also the double last names produce a lot of long names. A pair of long names that differ by a character or two will get a higher comparator score than a pair of short name differing by one or two characters.

5.2 The Edit Distance String Comparators

As discussed above, the coherence edit distance comparator is supposed to supplement the basic Levenshtein edit distance comparator by considering character transpositions. However, we found no evidence of it working better than the basic edit distance comparator on our data sets in terms of average sensitivity for given selectivity rates. The sensitivity values for the two comparators are similar with the coherence comparator sensitivity values generally less than the edit distance values. The only exception occurs in the problem three last name files, where the coherence edit sensitivity is slightly higher than the edit distance sensitivity, but in this case both are considerably below the Jaro-Winkler comparator sensitivities.

The longest common subsequence comparator was not thought to be a serious competitor, and indeed it substantially underperforms all of the other comparators on the relevant selectivity regions for these data sets. However, we did wish to consider the combined edit/lcs and coherence/lcs comparators. We look at the performance of the these two comparators along with the standard JW111 comparator for reference. For the 2000 first name pairs, the average sensitivities are given in Fig. 7. The LCSLEV comparator is close to the JW111 comparator, starting out somewhat below and then catching up, while the LCSCOH comparator has similar trajectory, but is considerably below the other two. The results for the 1990 names are similar. For the first 21 sets of 2000 last names, the JW111 comparator is very slightly above the LCSLEV comparator in the positive weight range (Fig. 8). At around the zero weight range, they agree and for the rest of the negative weight range, the LCSLEV starts to surpass the JW111 (Fig. 9). The LCSCOH comparator starts out considerably below the other two, but increases to meet the JW111 comparator near the end of the negative weight range. In the case of the last three sets of last names (Fig. 10), the JW111 comparator underperforms both of the edit-type comparators, with the coherence edit distance still slightly below the simpler Levenshtein edit distance sensitivities.

To summarize, the comparator based on coherence edit distance and the longest common subsequence length never exceeds the performance of the comparator based on standard Levenshtein edit distance and longest common subsequence length. This latter comparator generally performs comparably to the

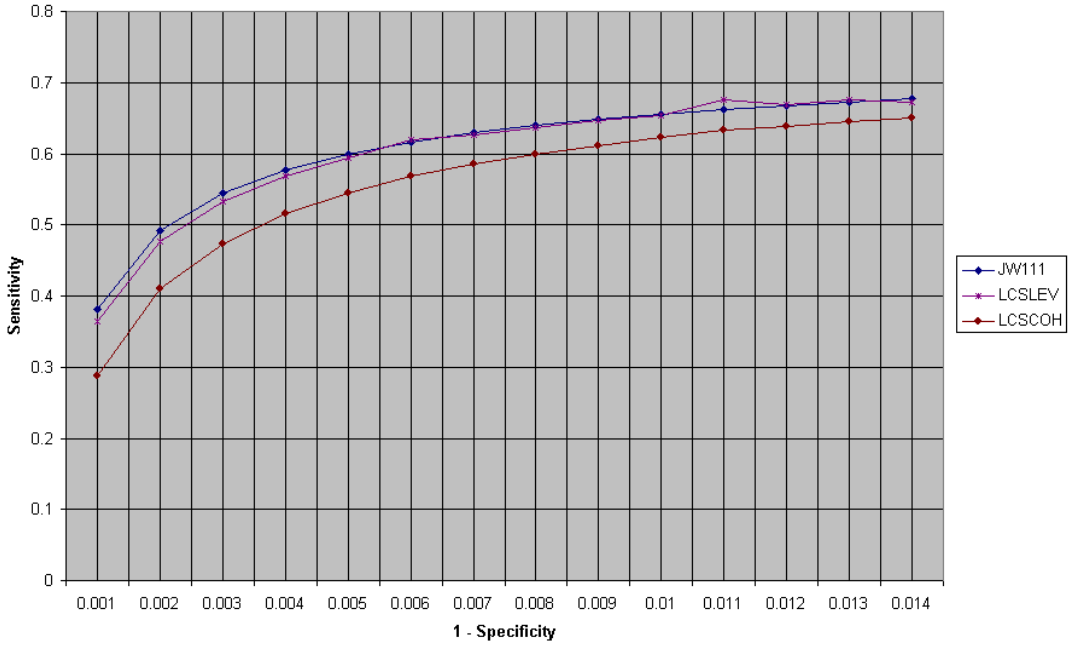

Figure 7: Average Sensitivities for LCSLEV, LCSCOH, and JW111 for 2000 First Names

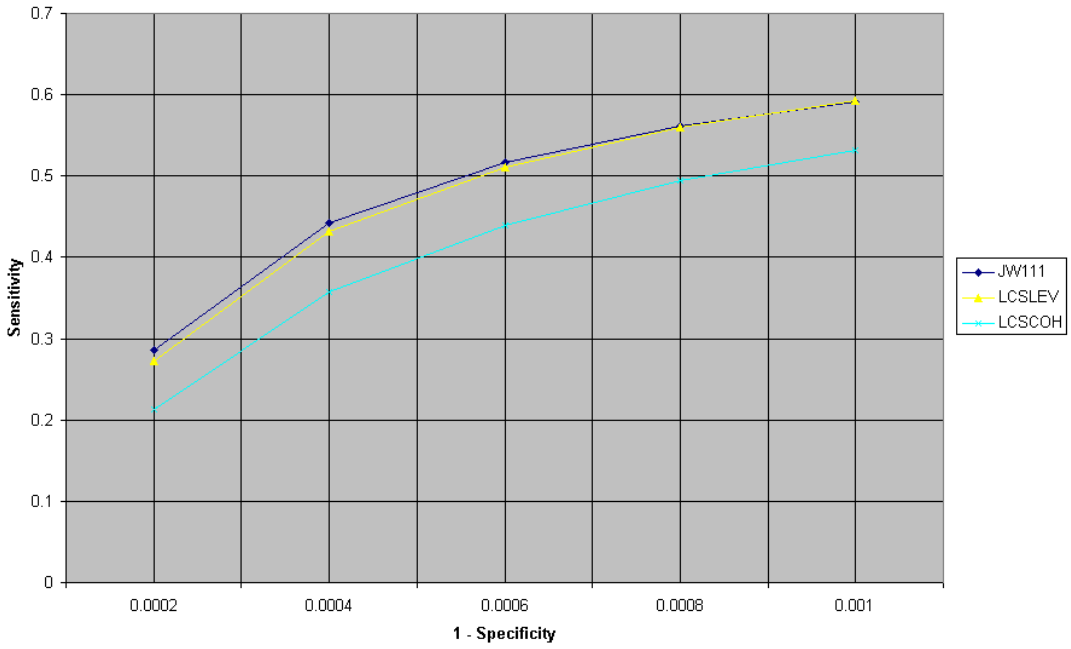

Figure 8: Positve Weight Range for Main Sets of 2000 Last Names for Edit Type Compartors

standard JW111 comparator. Usually the LCSLEV comparator is slightly lower in average sensitivity than that of the JW111 comparator in ranges corresponding to positive matching weights and slightly higher in ranges corresponding to negative matching weights. However, for our anomalous sets of last names, the LCSLEV comparator significantly exceeds the performance of the JW111 comparator.

5.3 Hybrid Comparator

5.3.1 Differences Between the Jaro-Winkler and Edit Distance Type Comparators

To understand how the performance of edit string comparators differs from that of the Jaro-Winkler comparators, we chose a selectivity cutoff value and looked at the pairs of names from matching records that are above the cutoff value for one comparator and below the cutoff value for the other comparator. We similarly looked at the analogous name pairs from non-matching records. Looking at the name pairs which have an above cutoff value for the J-W comparator and below cutoff value for the LCSLEV comparator, we did not perceive a pattern

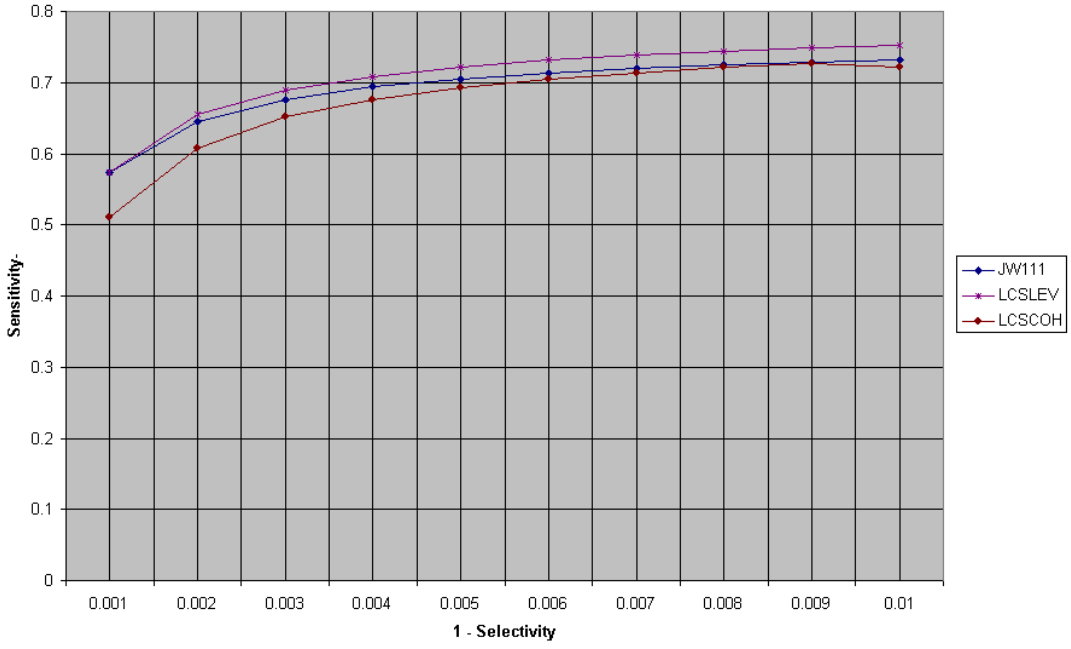

Figure 9: Full Weight Range of LCSLEV and LCSCOH Average Sensitivities for Main Last Name Sets

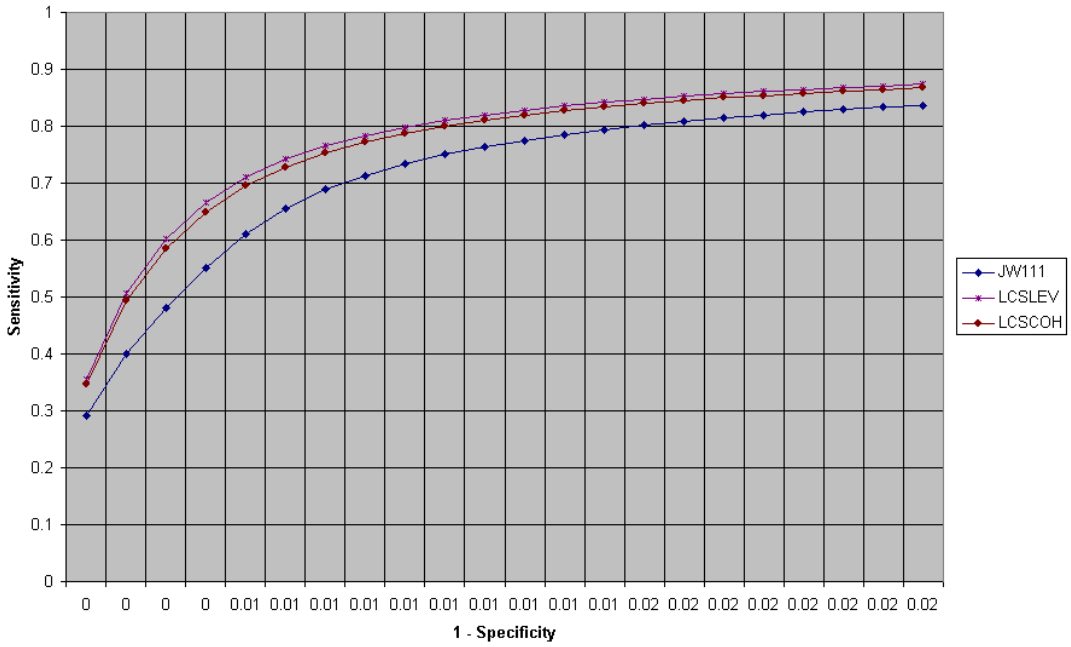

Figure 10: Last Three Last Name Sets for LCSLEV and LCSCOH

to the name pairs from either matching or non-matching records. However, there was a similarity to the name pairs that exceeded the LCSLEV cutoff and were below the J-W cutoff. These name pairs highlight some asymmetries of the J-W comparator that can result in some low scores especially for double names.

We can understand the major asymmetry of the J-W comparator as follows. Let α, β be two strings both of length n with no characters in common and let $\alpha\beta$ be the concatenation of these two strings. If we use the J-W comparator on the pair $\alpha, \alpha\beta$, we get a fairly high score. Specifically, the two strings have n common characters with no transpositions for a basic J-W score of

$$s = \frac{1}{3}\left(\frac{n}{n} + \frac{n}{2n} + \frac{n}{n}\right) = \frac{5}{6}.$$

If $n \geq 4$, the prefix adjustment raises the score to

$$s = \frac{5}{6} + \frac{4}{10}\left(\frac{1}{6}\right) = \frac{9}{10}$$

We might expect this comparator score to result in a comparison weight somewhere near 0, perhaps slightly positive. On the other hand, if we compare the strings $\beta, \alpha\beta$, the J-W comparator finds no common characters, since the search window for common characters has radius

$$r = \frac{2n}{n} - 1 = n - 1.$$

Actually, for a basic score below 0.7, the comparator does not bother to compute the J-W score adjustments, but in any case, we clearly should end up with a comparator score resulting in a full disagreement weight. On the other hand, the LCSLEV comparator results in the same score for either pair $\alpha, \alpha\beta$ or $\beta, \alpha\beta$. The string transformation requires n insertion/deletion edits and the longest common subsequence has length n. Thus the edit distance score is $1 - \frac{n}{2n} = \frac{1}{2}$ and the LCS score is $\frac{n}{n} = 1$, which averages to a combined LCSLEV score of

$$s = \frac{1}{2}\left(\frac{1}{2} + 1\right) = \frac{3}{4}.$$

The three last files of last name pair from 2000 especially contain a lot of Hispanic names where the surname from one file is reported as a double name and the other file just has one of the two names, so this distinction in comparator behavior can be relevant.

Another anomaly for the J-W comparator can occur when a common character occurs more than once. Since the common character search proceeds from left to right within the search window, this can result in a high transposition count. An example is the pair (*sara,asara*). The four common characters are (*s,a,r,a*) and (*a,s,a,r*), which counts as two transpositions. The resulting score is

$$s = \frac{1}{3}\left(\frac{4}{4} + \frac{4}{5} + \frac{2}{4}\right) = \frac{23}{30} \doteq 0.767.$$

There are no remaining similar characters and no agreeing prefixes. The strings are too short for the suffix adjustment. A JW111 score of 0.767 results in a full disagreement weight. On the other hand, the transformation from one string to the other costs one edit and the longest common substring has length 4, so the LCSLEV score is

$$s = \frac{1}{2}\left(\frac{4}{5} + \frac{4}{4}\right) = \frac{9}{10}.$$

The fact that the J-W algorithm does not always find the minimum number of transpositions for the common characters may have a modest role in distinguishing the performance of the two types of string comparators.

5.3.2 Selecting a Hybrid Comparator

The standard J-W comparator generally does well in our selectivity, average sensitivity analysis. The combination Levenshtein distance and longest common substring comparator performs comparably. However, in our extreme cases of the last three sets of the last name pairs, the edit distance type comparator does significantly better than the J-W comparator, probably because of the more robust handling of the single name/double name pairs. We consider that it might be advisable to use the J-W comparator except in those cases where it gives a small value compared to the edit comparator. However, we need to be able to compare the string comparator values from the J-W comparator and the edit distance comparator.

We tried combining the JW110 comparator with the LCSLEV comparator. We used the one without the suffix adjustment since this adjustment generally made a small difference, sometimes this difference was negative, and we thought of the LCSLEV comparator as offering a suffix correction. We considered the values of the JW110 and LCSLEV comparators for the same selectivity values, restricted to selectivity values in a range relevant to the assignment of varying matching weights. The comparators show a strong and consistent linear relationship in this range, where we estimated the regression coefficients to be

$$\hat{s}_{110} = 0.66 s_{lcslev} + 0.38$$

and define the hybrid string comparator score by

$$s_h = \max\left(s_{110}, \min\left(\hat{s}_{110}, 1\right)\right).$$

This may not be the best way to combine the two string comparators. This hybrid string comparator has some unappealing formal properties. The minimum value of s_h is 0.38 instead of 0, but comparator values this low will be assigned the full disagreement weight anyway. Also it is possible to have $s_h(\alpha, \beta) = 1$, but $\alpha \neq \beta$ For instance, if $\alpha = a_1a_2a_3a_4a_5a_6a_7a_8a_9$ has distinct characters and $\beta = a_2a_3a_4a_5a_6a_7a_8a_9$, then

$$s_{lcslev} = \frac{1}{2}\left(\frac{8}{9} + \frac{8}{8}\right) = \frac{17}{18} \doteq 0.944$$

which results in $\hat{s}_{110} > 10$. On the other hand, we have

$$s_{110} = \frac{1}{3}\left(\frac{8}{9} + \frac{8}{8} + \frac{8}{8}\right) = \frac{26}{27}$$

and

$$s_{111} = \frac{26}{27} + \frac{1}{27}\left(\frac{8-1}{8+9+2}\right) = \frac{167}{171} \doteq 0.97660.$$

This is a high score but it would receive somewhat less than the full agreement weight.

5.3.3 Assessing the Hybrid Comparator

The selectivity/average sensitivity values for the positive weight range for the 2000 first name pairs are shown in Fig. 11 and for the full weight range in Fig. 12. We see that in the highest selectivities, the hybrid comparator JWLEV2 has average sensitivities very close to (within 0.01) the standard J-W comparator and distinctly above the edit/lcs comparator. In the early stages it is very slightly below the JW111 comparator, but catches up and by the negative weight range, the sensitivities exceed (by more than 0.01) those of the J-W comparator. The results for the 1990 names files look very similar. The results for the standard last name files from 2000 are shown in Fig. 13 and for the full weight range in Fig. 14. The positive weight range looks similar to that for the first names, except that toward the end the hybrid comparator sensitivities exceed (by at least 0.01) those of JW111 rather than just catching up. In the full weight range, the JW111 sensitivities are exceeded by the edit/lcs sensitivities, but the hybrid comparator sensitivities exceed both, exceeding the J-W comparator by 0.03. For the anomalous three last name sets, we see in Fig. 15 that the edit/lcs comparator well exceeds the J-W comparator and that the hybrid comparator is close to the edit/lcs comparator. Where the hybrid comparator average sensitivity values are less than those of the LCSLEV comparator by between 0.02 and 0.03, they exceed those of the JW111 comparator by between 0.08 and 0.09.

5.4 Summary

The Jaro-Winkler string comparators perform similarly with the three adjustments. The prefix adjustment always helps, the similar character usually helps modestly, and the suffix adjustment generally either helps or hurts a small amount. The adjustments do the most good in the higher score ranges, boosting the scores of already similar string pairs. They have less effect with less similar pairs, when the similar character and suffix adjustments begin to lower rather than raise sensitivities.

The Levenshtein edit distance and coherence edit distance metrics can be effective when combined with the longest common substring score. However, the coherence comparator always does less well than the basic edit distance

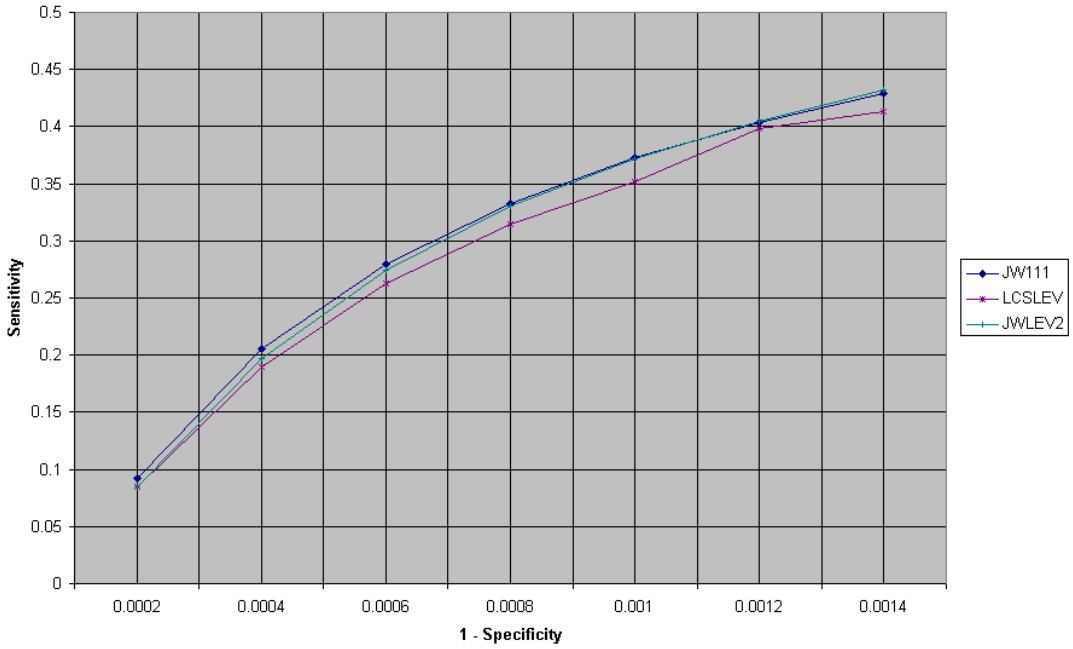

Figure 11: Average Sensitivities for Positive Weight First Names for Hybrid Comparator

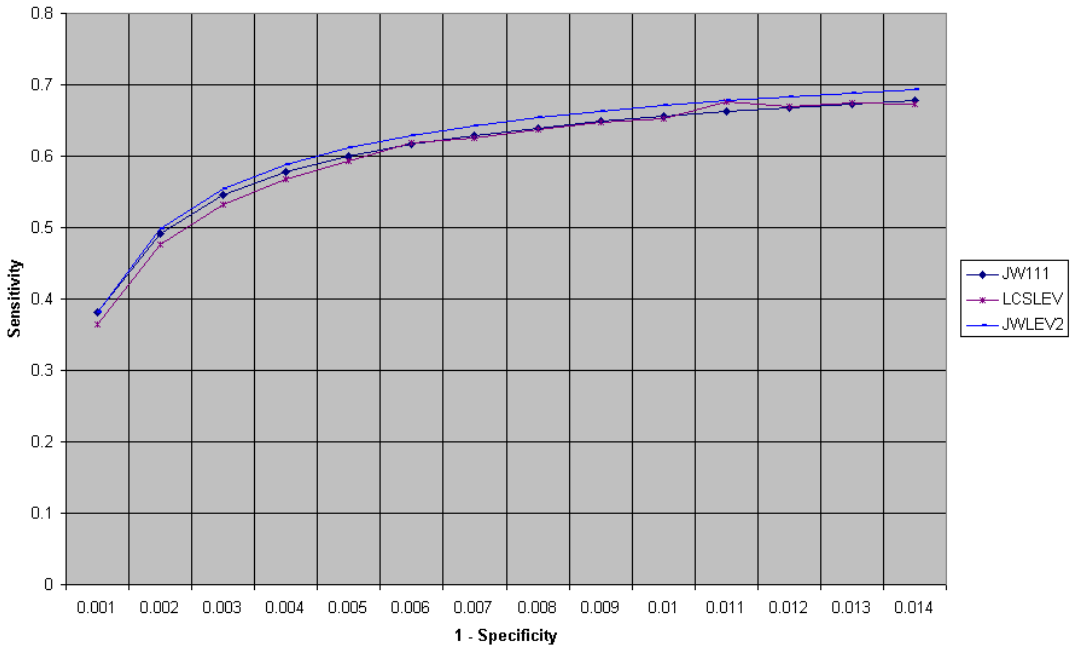

Figure 12: Average Sensitivities for 2000 First Names for Hybrid Comparator

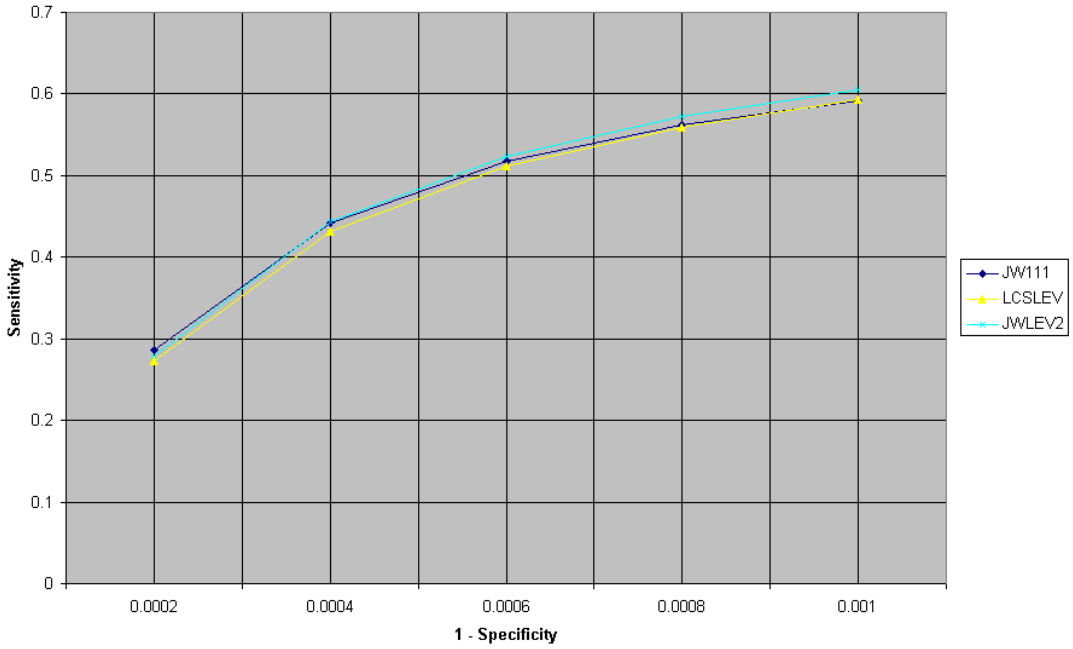

Figure 13: Average Sensitivities for Positive Weight Range of Last Name Pairs for Hybrid Comparator

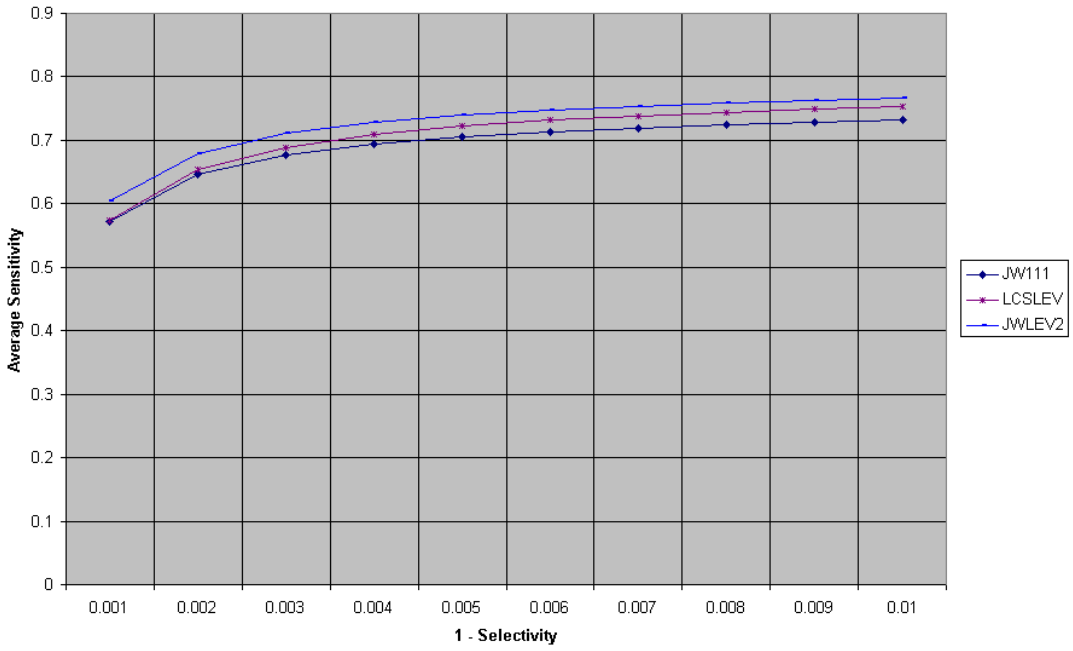

Figure 14: Average Sensitivities for 2000 Last Names for Hybrid Comparator

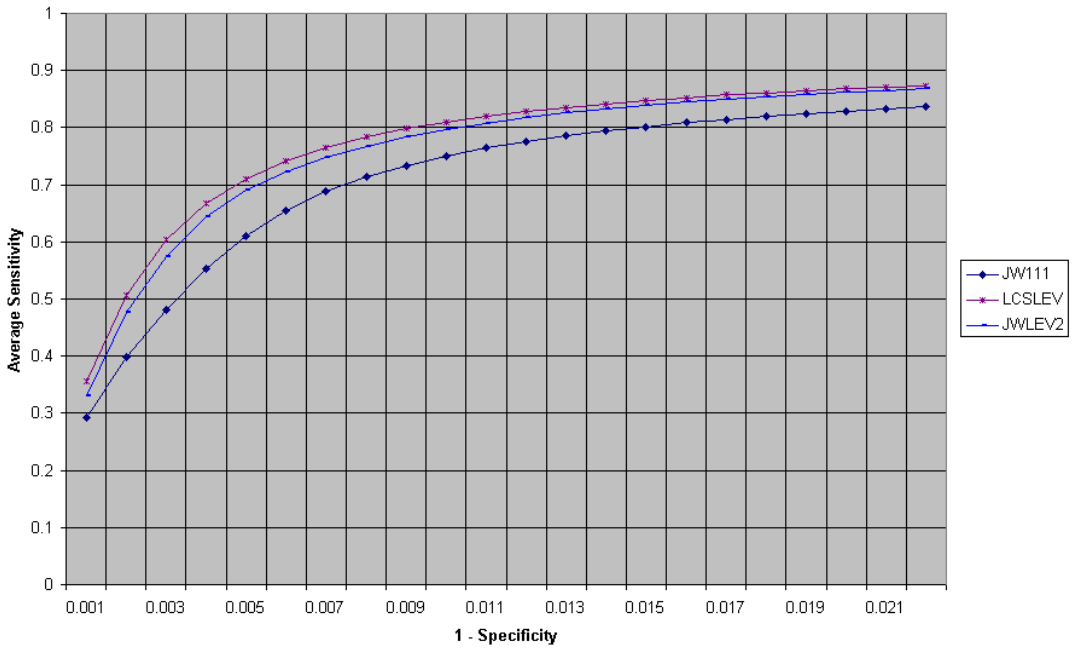

Figure 15: Sensitivities for Difficult Last Names for Hybrid Comparator

comparator, so there does not seem to be any justification for the extra complexity for this application. One might experiment with different coherence index other than $N = 4$, but this does not appear to be very promising.

The hybrid string comparator formed from the maximization of the J-W comparator JW110 and the scaled edit/lcs comparator does as well as and sometimes better than the standard JW111 comparator. It shows more robustness in our samples, doing about the same as JW111 for more similar pairs and doing better for more problematic pairs. Specifically it does a lot better with the sets of long Hispanic double last names. Of course, even if this enhanced performance persists in other examples, it does have the cost of extra computational complexity, taking about three times as long to compute at the standard Jaro-Winkler comparator.

6 Matching Results

We have analyzed several string comparators by examining their average sensitivity over selectivity intervals, concentrating on intervals where the string comparator values can have some influence on the matching weights for record pairs in record linkage. We have seen some, mostly slight, differences between these comparators. We would like to know if these measured differences are enough to effect the final record linkage result.

6.1 Using Bigmatch on the 2000 Data

We first ran Bigmatch for the 2000 Census/ACE files, first using the standard Jaro-Winkler comparator with all three options and then with the hybrid Jaro-Winkler and Levenshtein distance/LCS comparator. We used three blocking passes: cluster number and first character of last name, cluster number and first character of first name, and cluster number where we used first and last name inversion for matching computation. We cut off the output at matching weight 0. We examined the output pairs sorted by decreasing matching weight. We note that the Bigmatch program does not provide one-to-one matching, although the files have 606,412 matching pairs.

The first blocking pass produces the bulk of the output pairs. When we try to compare the results of the J-W comparator and the hybrid comparator, the results are not conclusive. If we compare the number of matched pairs as a function of the number of false match pairs, then the two outputs go back and forth between which has the larger number of true matches for a given level of false matches. After some initial volatility, the ranges settle down to a difference of a few hundred matches either way, with the hybrid comparator matcher generally averaging about 200 more matches. If we consider average number of matches for a given level of false matches, it is difficult to tell the two outputs apart. One way that this output could be used is to decide on a cutoff matching weight level and take the links above this level as designated matches. If we look at the number of false matches above a given weight, we see that

Cutoff	Weight	Description
A	13.5775	Disagree on first name, a missing middle initial
B	11.6947	Disagree on first name and middle initial
C	10.7901	Disagree on first name and sex, a missing middle initial
D	9.1807	Disagree on first name and sex, missing a middle initial and relationship to head of household

Table 6: Possible Matching Weight Cutoff Points

Cutoff	Matches	Non-Matches	New Matches	New Non-Matches
A	432,992	439	1139	512
B	486,915	1975	87	113
C	505,736	2475	38	237
D	533,690	3808	45	150

Table 7: J-W Output at Cutoffs

there are a few weights where a relatively large number of false matches enter. As it turns out, these matching weights are the same for both outputs since they are not influenced by string comparator values. We could consider these points as possible cutoff values. The points are described in Table 6, where the records agree on all fields except the ones listed. In Table 7 we list the total number of matches and non-matches above the cutoff point and the number of new matches and non-matches that are included at this cutoff value for the version of Bigmatch using the standard Jaro-Winkler comparator. In Table 8 we do the same for the output of Bigmatch using the hybrid string comparator. There is some indication that the hybrid comparator is doing slightly better in that at these levels it allows in a few more false matches (from 13 to 25) which it has a larger number of true matches (initially almost 3000, settling down to over 1000). Of course, from the point of view of the total number of matches, these differences are a small proportion.

The second blocking pass used cluster number and first character of first name. We sorted each output by matching weight, then accumulate the counts of true and false matches for decreasing matching weights. We plot in Fig. 16 the number of true matches for a given number of false matches at the same matching weight. The program using the hybrid matcher shows more matches for a given number of false matches, generally averaging around 200

Cutoff	Matches	Non-Matches	New Matches	New Non-Matches
A	435,935	452	1026	514
B	488,696	1977	88	116
C	506,799	2500	35	236
D	534,797	3831	38	150

Table 8: Hybird Output at Cutoffs

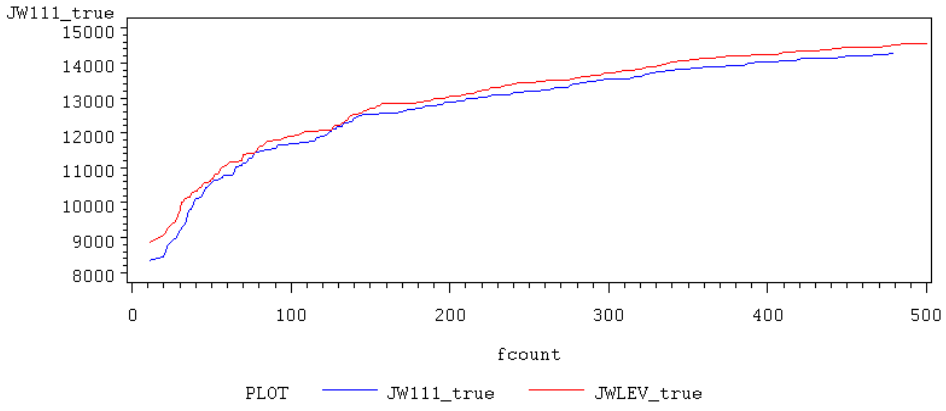

Figure 16: Number of True Matches for Given False Matches Using JW111 and JWLEV

more matches. As a fraction of the total number of matches found, this is very small, so the match rate is not much changed. However, this consistent excess is probably due to differences of string comparator evaluation for last names, specifically double Hispanic last names.

By the third blocking pass, most of the matching record pairs have already been culled out. However, the first and last name inversion does find some new matches. Depending on the number of false matches that are tolerated for a cutoff value, between about 3300 to 3700 new matches are found. At the same level of false matches in this range, the hybrid comparator version finds about 40 more matches than the standard comparator version.

6.2 Using the One-to-One Matcher on the 2000 Data

One use of the Bigmatch program is to extract a file of likely matching records from one of the two files at hand, so that the reduced file can be used with the SRD Matcher to extract one-to-one matches. Since the Census and ACE files have already been clerically matched one-to-one, using a one-to-one matcher seems to be in order. We used the Bigmatch program twice, first extracting subfiles of likely matches from one file, then extracting subfiles of likely matches from the second file. We then use the one-to-one matcher on these pairs of reduced files.

The largest files come from the first blocking pass which used cluster number and first character of last name. The true matches against false matches for the two outputs is shown in Fig. 17. Since the records have already been selected as likely matches, we see that the scale of the true matches is very large compared to the scale of the false matches. Furthermore, while the initial rate of true matches to false matches slows down, there is a significant increase in

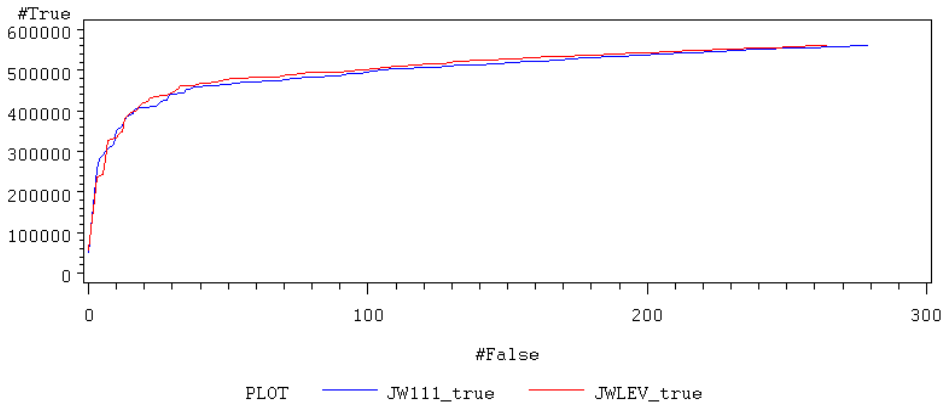

Figure 17: One-to-one Matching First Blocking Pass

true matches throughout. Comparing the two outputs, we see after some initial instability, the hybrid comparator consistently has more true matches per false matches throughout, although the difference gets smaller as we proceed farther along. In general with matching results, we would choose a cutoff point above which we accept all of the pairs as valid links and another cutoff point below which we assume all of the pairs are false links. However, since these sets are so rich in matches, we would probably not have a lower cutoff value and would accept everything as at least in the clerical region. If we accept the whole set as links, then there is not much difference between the sets. The J-W comparator set has 560,556 true matches and 279 false matches and the hybrid set has 560,571 true matches and 264 false matches, a difference of just 17 record pairs. On the other hand, if we chose our high cutoff value to include only the rapidly rising true match region, we might comparably choose a cutoff at around 38 false matches. If so, then the J-W set would have 458,835 true matches and the hybrid comparator set would have 463,378 true matches. If we take the rest of the sets to be (rather large) clerical regions, then the two clerical regions have very similar proportions of false matches and the hybrid comparator set has 4543 fewer records.

The second set of files result from blocking on cluster number and first character of first name. These records are those with high matching scores that have not already been collected in the first blocking pass sets. We see the true/false matching values in Fig. 18. In this case, the hybrid true matches more clearly exceed the J-W true matches throughout, although they come close together at the end. If we again accept the complete sets as designated links, then there is not much difference between them. The J-W set has 15,007 true matches and 310 false matches, while the hybrid set has 15,013 true matches and 304 false matches. However, if we choose a high cutoff value for the region of rapid true match increase, then we might compare the two at 16 false matches,

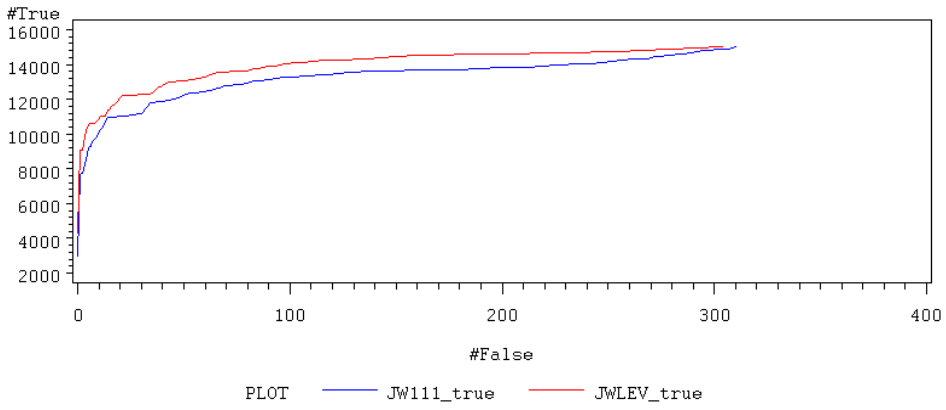

Figure 18: One-to-one Matching Second Blocking Pass

where the J-W set has 10,969 true matches and the hybrid set has 15,567 true matches. If we take the remained of the sets as clerical regions, then the hybrid set has 598 fewer records, about a 13.8% reduction in the size of the clerical region.

The third set of files result from blocking on just cluster number and using comparing the names using first and last name inversion. The true/false graph is given in Fig. 19. The shape of the graph is not typical, probably because the sets represent the residual record pairs not identified by the previous two blocking passes. We used name inversion to try to pull out a few extra matches. However, we see that the true matches never rise rapidly with respect to the false matches, so one possibility is to regard the whole set as the clerical region. In this case, the two outputs are similar, with the J-W comparator producing 3877 true matches and 84 false matches and the hybrid set producing 3881 true matches and 82 false matches. If we wish to designate an upper cutoff, then we see that it depends how high we choose to make it. Initially the hybrid true count is a little above the J-W true count, but then the J-W true count exceeds for most of the way. For example, if we choose a false match level of 15, then we have 535 matches from the J-W comparator and 446 matches from the hybrid comparator. The resulting clerical review regions have the J-W region reduced by about 2.6% from the hybrid region.

6.3 Using the One-to-One Matcher on the 1990 Data

The 1990 data is somewhat different from the 2000 data. In addition to being smaller sets, the records in one set do not necessarily have a match in the other set. Thus it is likely that one would choose a lower cutoff as well as an upper cutoff. We see the output of the 2021 set in Fig. 20. Again there is not much difference in the total number of true matches in the two sets, the J-W having

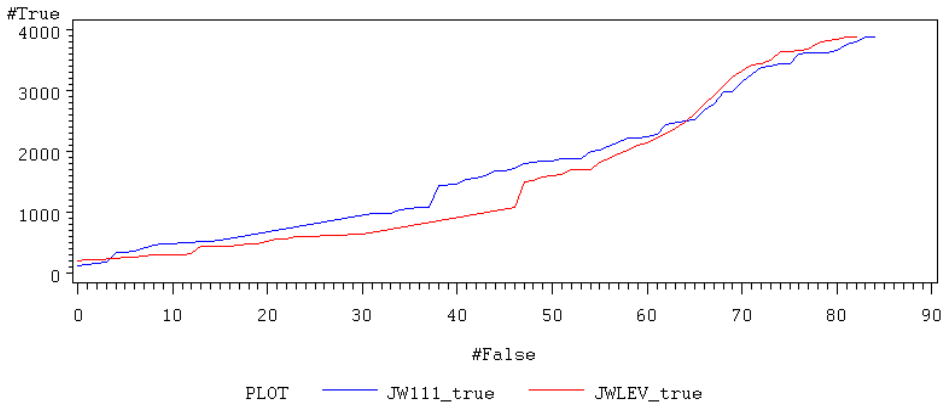

Figure 19: One-to-one Matching Third Blocking Pass

3417 matches and the hybrid having 3419 matches, but the hybrid curve stays above the J-W curve throughout. If we choose a high cutoff near the top of the steep part of the curve, we might compare the results at 27 false matches. Here the J-W matcher has 3318 true matches and the hybrid matcher has 3359 matches. If we choose a low cut at 83 false matches where the curves level out, the total number of matches above the low cutoff is still close (3416 and 3417 respectively), but the clerical region with 56 false matches has 98 true matches for the J-W set and 58 true matches for the hybrid set, a 26% reduction in the size of the clerical region.

The true/false graph for the 3031 data set is shown in Fig. 21. The hybrid curve is on top but the graphs are closer. Again the total number of matches is similar, 3547 for the J-W matcher and 3549 for the hybrid matcher. If we take the high cutoff around the top of the steep part, we can choose a level of 13 false matches to compare, with the J-W matcher having 3442 matches and the hybrid having 3473. If we take a low cutoff at 47 false matches, the clerical region has 34 false matches, the J-W output has 89 true matches, and the hybrid matcher has 68 true matches. The result is that the hybrid matcher has a total of 10 more matches in the two regions and a clerical region reduced by 17%.

The graph for the STL data set is given in Fig. 22. The hybrid curve is more separated above the J-W curve, similarly to the 2021 data set case. As usual, the total number of matches in the output sets is similar, 9860 for the J-W comparator and 9863 for the hybrid comparator. If we choose a high cutoff at 39 false matches, then the J-W matcher has 9712 matches and the hybrid matcher has 9785 matches above the high cutoff. If we take the low cutoff at 146 false matches, the J-W matcher has 9856 matches and the hybrid matcher has 9859 matches above the low cutoff. This makes the clerical region have 105 false matches, the J-W matcher has 144 matches and the hybrid matcher has 74 matches for a 28% reduction in the size of the clerical region.

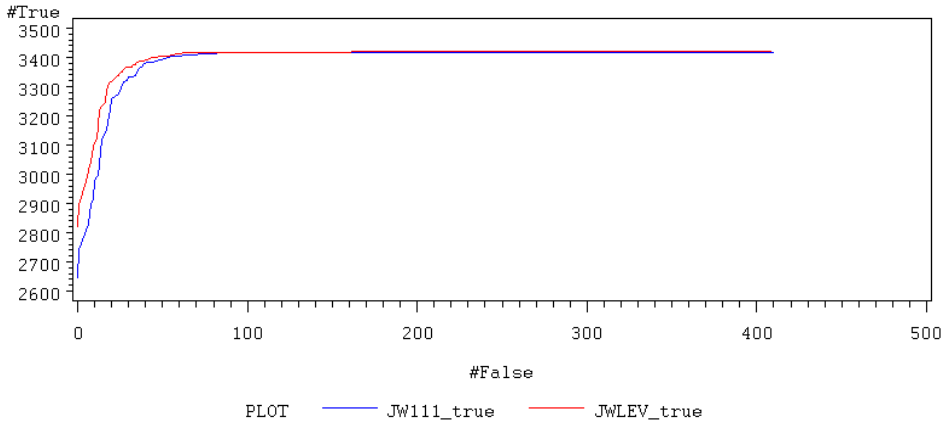

Figure 20: Matching the 2021 File

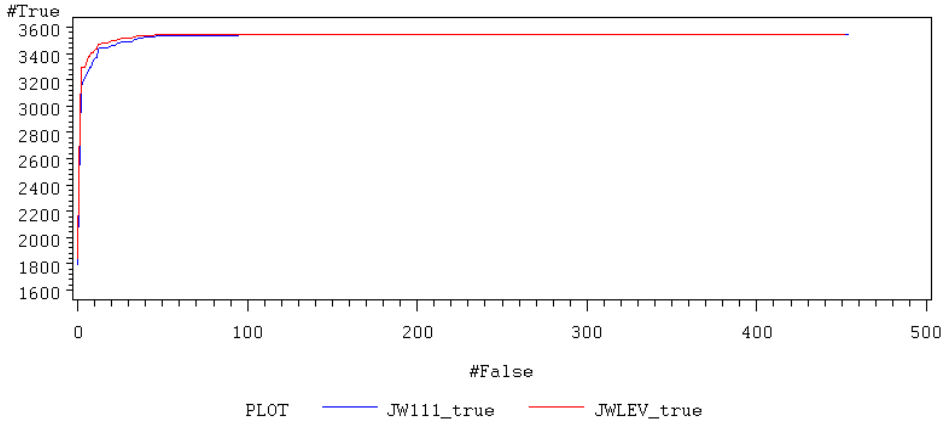

Figure 21: Match Results for 3031 File

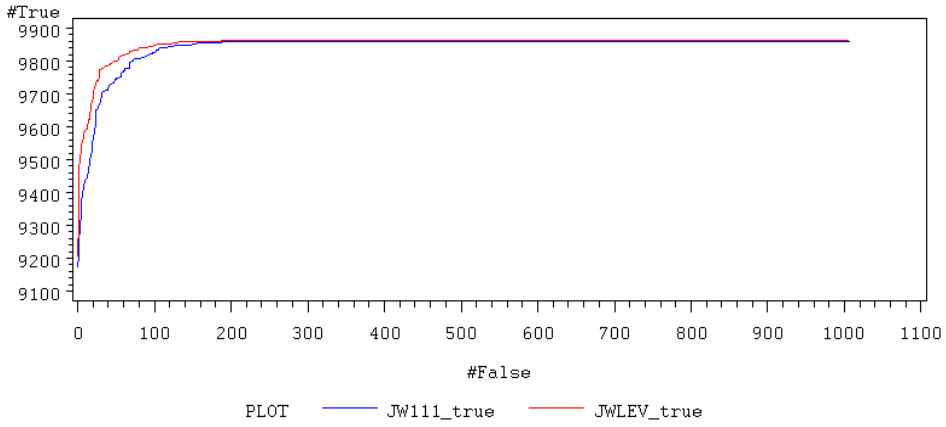

Figure 22: Match Results for STL File

6.4 Summary

In the previous analysis using ROC curve values, we saw that the Jaro-Winkler comparator with all three adjustments and the hybrid comparator which combines the Jaro-Winkler comparator without the suffix adjustment and the combination of edit distance and longest common subsequence comparator both were good performers in classifying the Census name typographical error data. The hybrid comparator appeared to generally do somewhat better. When we use the two comparators in our matching software, the hybrid comparator continues to do slightly better in classifying the matches and the non-matches. It finds very few extra matches, but it does tend to separate the matches from the non-matches. The cost is that the hybrid matcher takes longer to run, essentially performing three quadratic algorithms instead of one.

References

[1] Stephen, Graham A. *String Searching Algorithms.* World Scientific Publishing Co. Pte. Ltd., 1994.

[2] J. Wei. "Markov Edit Distance". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 26, No. 3, pp. 311–321, 2004.

[3] Winkler, William E. "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage". *Proceedings of the Section on Survey Research Methods, American Statistical Association*, 1990, pp. 354–359.

[4] Zou, Kelly H. *Receiver Operating Characteristic (ROC) Literature Research.* http://splweb.bwh.harvard.edu:8000/pages/ppl/zou/roc.html.