Paper 1674-2014

Matching Rules: Too Loose, Too Tight, or Just Right?

Richard Cadieux, Towers Watson, Arlington, VA & Daniel R. Bretheim, Towers Watson, Arlington, VA

ABSTRACT

This paper describes a technique for calibrating street address match logic to maximize the match rate without introducing excessive erroneous matching.

BACKGROUND

Look up tasks that involve matching street address information between a source and a look up table can be very onerous, due to the significant potential for variation in the way many street addresses can be represented. For example, how many variations of "First Crown Pt Road" might you encounter?

- First Crown Point Rd
- First Crown Point Road
- 1st Crown Pt Road
- 1st Crown Pt Rd
- etc.

Thus, we need a method of reconciling the representation of source address data (which in our case will vary with each individual client) to our look up table of street address with nine digit ZIP Code, which we obtain from the US Postal Service and is generally formatted according to their standards.

The challenge is to allow for matches that are not perfect, which we refer to as "fuzzy matches", while minimizing spurious matches that would yield erroneous look ups.

Before looking for fuzzy matches, we looked for exact matches for the street name and street number. The original address data is one string with all parts of the address except city, state, and ZIP Code. We split this address into sections for matching to the look up data. We created variables identifying street number, street name, street suffix, street pre-direction, street post-direction, secondary type and secondary number. Once street name was identified, we compressed spaces and most special characters and set all characters to uppercase. Certain words that have a predictable range of values were set to a standard value in both the original address and the lookup data. For example, FIRST is changed to 1st.

Key SAS Statements

- COMPGED function – Returns the generalized edit distance between two strings. Generalized edit distance is a generalization of Levenshtein edit distance, which is a measure of dissimilarity between two strings. The Levenshtein edit distance is the number of deletions, insertions, or replacements of single characters that are required to transform *string-1* into *string-2*.
  - Syntax – COMPGED(*string-1*, *string-2* <,*cutoff*> <,*modifiers*>)
  - Returns values that are multiples of 10, e.g., 20, 100, 200
  - A match returns a value of 0.
  - We created the variable COMPGED_score based on the COMPGED function.

- SPEDIS function – **Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words.**
    - Syntax – **SPEDIS**(*query*,*keyword*)
    - Returns values in increments of 1, usually less than 100
    - A match returns a value of 0.
    - The value is dependent on the length of the query word.
    - We created the variable SPEDIS_score based on the SPEDIS function.

APPROACH

We originally attempted to use the COMPGED function to identify fuzzy matches. The COMPGED function is the edit distance between two strings, assigning a score for each edit function needed to make the 2 strings match. The strings must still match by street number and ZIP Code. We created the variable COMPGED_score based on the score from the COMPGED function. Using a cap of 590 on the COMPGED_score, we created a fuzzy match for a portion of the records that were not successfully matched using "exact match" rules.

The first problem with these results was that there were some records where the matched name was subjectively incorrect, or that the matches were too fuzzy. Most of these incorrect matches were in the higher COMPGED_scores, but even lowering the cap to 290 there were still some incorrect matches, and then we started to exclude matches that were objectively correct (see Exhibits 2 and 3)

At this point, we introduced a second function, SPEDIS. The SPEDIS function determines how likely it is that two words match, based on the spelling distance. A score is assigned on the changes needed to make the two words match. We created the variable SPEDIS_score based on the SPEDIS function.

We wanted to assign a cap on the SPEDIS_score, but we decided a single cap would be too fuzzy also. We create five thresholds based on looking at sample records in different ranges of scores. The five groups were created based on the COMPGED_score, then the SPEDIS_score was calibrated until we thought we captured the most matches with minimal, but not zero, incorrect matches.

Determining the proper cap is a subjective process. There are valid matches that a human can see that the program will not score within the thresholds, and there are invalid matches that the program will assign low scores, a situation that is illustrated in some of the examples below. The threshold could change depending on what kind of results can be tolerated and what kind of data is being matched. This is where programmers need to use their knowledge of the data and what the client needs to determine thresholds for each project.

Looking at the sample records for adjusting the SPEDIS_score revealed some values that could be standardized in the step before matching and avoids the fuzzy matching for those addresses.
One option that we explored was to set the SPEDIS_scores low enough to exclude all incorrect matches (see Exhibit 4). Instead we calibrated the SPEDIS_score and COMPGED_score to the current makeup, which increases the number of correct matches while allowing a small number of incorrect matches (see Exhibit 5).

The first threshold was COMPGED_score 10 to 100 and SPEDIS_score of 100 or less. Most COMPGED_scores of 100 or less are valid matches. Only one letter can be inserted inside a word but multiple letters can be added at the end of a word. The SPEDIS_score of 100 catches the few matches where the original word is one or two letters and the match adds multiple letters. More than 99% of records with a COMPGED_score of 10 to 100 had a SPEDIS_score of 100 or less. See Figure 1 for the results.

Acceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
WINDCHESTER	WINCHESTER	100	9
HIGHLANDSPIRNGS	HIGHLANDSPRING	70	5
8TH	8THAVENUE	60	100
COMM	COMMONWEALTH	80	100

Unacceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
C	C478	30	150
S	SPENCE	50	250

Acceptable Matches but Outside Acceptable Score

Original	ZipMatch	COMPGED	SPEDIS
FL	FLORIDA	50	125

Figure 1. First Threshold: COMPGED_score from 10 to 100 and SPEDIS_score <= 100.

The second threshold was COMPGED_score 110 to 190 and SPEDIS_score less than 50. Matches with a COMPGED_score from 110 to 190 usually have one letter inserted in the middle and multiple letters added to the end of the word. We used a lower SPEDIS_score for these matches because there were more erroneous matches. About 80% of records with a COMPGED_score of 110 to 190 had a SPEDIS_score of less than 50. See Figure 2 for the results.

Acceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
31TERR	31STTERRACE	150	37
AUGUSTAPLTN	AUGUSTAPLANTATION	160	27

Unacceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
STAR	SHARON	120	50
WWOOD	WOODCRAFT	160	60

Acceptable Matches but Outside Acceptable Score

Original	ZipMatch	COMPGED	SPEDIS
ROW3	ROWTHREE	140	75
CRU	COUNTYROADU	180	133

Figure 2. Second Threshold: COMPGED_score from 110 to 190 and SPEDIS_score < 50.

The next threshold was COMPGED_score 200 to 290 and SPEDIS_score 30 or less. Matches with a COMPGED_score from 200 to 290 include changes to the first letter of the original word. There are also matches with multiple changes to other letters. We lowered the SPEDIS_scores for these matches to 30 or less. About 50% of records with a COMPGED_score from 200 to 290 have a SPEDIS_score of 30 or less. See Figure 3 for the results.

Acceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
INDIANMDS	INDIANMOUND	220	27
BELVERDE	BELVEDERE	210	25

Unacceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
YONKER	YANKEE	200	33
CHAMBLE	CHANDLER	210	35

Acceptable Matches but Outside Acceptable Score

Original	ZipMatch	COMPGED	SPEDIS
MORWAY	NORWAY	200	33
EDENBERG	EDINBURGH	210	31

Figure 3. Third Threshold: COMPGED_score from 200 to 290 and SPEDIS_score <= 30.

The fourth threshold was COMPGED_score 300 to 390 and SPEDIS_score less than 30. The maximum score for a single change in COMPGED_score is 200, so these matches always had at least two changes. We lowered the SPEDIS_score slightly for these matches to less than 30. About 17% of records with a COMPGED_score from 300 to 390 had a SPEDIS_score less than 30. See Figure 4 for results.

Acceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
MOUNTMORIAH	MTMORIAH	300	27
COWPENLKPT	COWPENLAKEPOINT	330	25

Unacceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
VALLEYBEND	VALLEYHEDGE	310	30
FORRESTRIDGE	FORESTHILL	370	30

Acceptable Matches but Outside Acceptable Score

Original	ZipMatch	COMPGED	SPEDIS
MOUNTOLIVE	MTOLIVE	300	30
GEOWASH	GEORGEWASHINGTON	360	64

Figure 4. Fourth Threshold: COMPGED_score from 300 to 390 and SPEDIS_score < 30.

The last threshold was COMPGED_score of 400 to 590 and SPEDIS_score 25 or less. The changes in this range are often incorrect because of the number of changes made to the original word. We only used if the SPEDIS_score was 25 or less, which accounted for less than 2% of records with a COMPGED_score from 400 to 590. See Figure 5 for results.

Acceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
OAKRIDGEPLNTN	OAKRIDGEPLANTATION	420	19
CDREJOSHUABARNEY	COMMODOREJOSHUABARNEY	500	15

Unacceptable Matches

Original	ZipMatch	COMPGED	SPEDIS
SAINTCLAIR	SAINTCASSIAN	410	35
CHESTERBEND	SHASTABEND	500	45

Acceptable Matches but Outside Acceptable Score

Original	ZipMatch	COMPGED	SPEDIS
KENNEDY	JFKENNEDY	400	28
CR1102	COUNTYROAD1102	580	66

Figure 5. Fifth Threshold: COMPGED_score of 400 to 590 and SPEDIS_score <= 25.

In the code (see Exhibit 1), the dataset 'unmatched_addresses' is joined with the dataset 'zipdata'. 'Unmatched_addresses' is all of the address records which did not match to an address in the zipdata in one of the previous 6 steps. The first limit is a COMPGED_score that is less than 600, and the match is by street suffix, ZIP Code, street number, and even-odd indictor for the street number.

The SPEDIS_score is calculated in the next step. There is also a check to find if there are differences in numbers between the two street names that were matched. We did not want to change numbers in an address because the fuzzy match was too random. We checked if the numbers in the original address were changed in the matching address. We did allow for numbers to be removed from the original address because there were cases where an apartment number was appended to the end in the client's street name and we did not identify the apartment number in the original processing of the address. For example, the original address of HILLSBORO MILE #308 is matching to the zip address of HILLSBORO MILE. The #308 was the apartment number but our rules did not recognize because the full address had NORTH after the apartment number, and there was no street suffix in the address. The fuzzy match sees HILLSBOROMILE#308 and HILLSBOROMILE and gave it a score that we can use (180 COMPGED_score and 5 SPEDIS_score).

The last two steps sort by the key, which is set earlier in the program and will be at least one member id variable, as well as nummatch flag, COMPGED_score, and SPEDIS_score and keep only the first record

for each key. There were often multiple matches for a street name. Then only the matches that fit within the ranges we determined were kept. COMPGED_score is always in multiples of 10, whereas SPEDIS_score has increments of one.

SUMMARY

The first step of our programming assignment was to define the goal of the process. The goal can range from producing a match for every record, no matter how fuzzy, to using very tight restrictions and only producing a few new matches. We chose to aim for producing more matches and including some unacceptable matches because we were looking for directional information and did not need to be perfect. We also did not need to include very fuzzy matches. Setting the goal is a subjective process that should be decided before calibration starts.

The calibration process is necessarily a process of trial and error. A key to making this process work for us was to start simple and add complexity after we understood the initial results. We used one metric, the COMPGED function, then analyzed the results of the matches to determine how many acceptable matches there were in each range of COMPGED_scores. We analyzed individual matches so that we knew the percentage of acceptable matches for each range of COMPGED_scores before moving on to the next step. When we decided to add a second function, we had a framework of understanding based on the COMPGED_score, which made it simpler to assign ranges for the SPEDIS_score.

Fuzzy matching can be used to increase the percentage of matches on address data. The programmer can calibrate the matching to include as many or as few matches needed depending on the tolerance of erroneous matches. This process can be adapted to other types of data where typos or non-standard abbreviations are common. The ability to customize matches is a valuable tool for programmers.

ACKNOWLEDGEMENTS

CONTACT INFORMATION

For additional information please contact:

Richard Cadieux
Towers Watson
901 North Glebe Road
Arlington, VA 22203
richard.cadieux@towerswatson.com

Exhibit 1

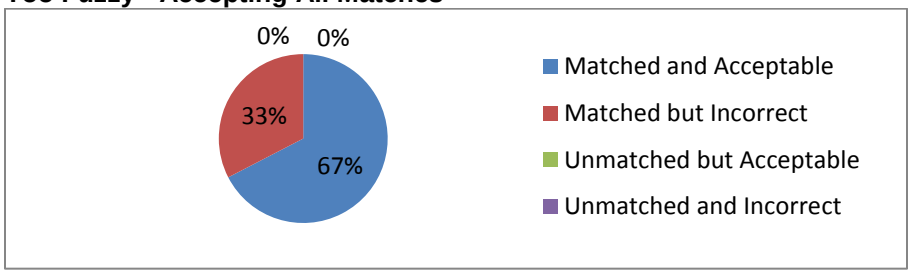
```
** Match records that fail all base checks and try to find closest street name match  **;
** Close matches may be street name misspellings in vendor file and can be matched    **;

proc sql;
   create table zipmerge as select
   a.*, COMPGED(a.street_name,b.street_name,600,'LN') as COMPGED_score, b.street_name as
street_name_zip,
   b.zip4_add_on_high, b.zip4_add_on_low, b.street_suffix as street_suffix_zip4,
b.street_predirection,
   b.street_postdirection,
   b.secondary_address_type, b.secondary_even_odd_both, b.secondary_high_address,
b.secondary_low_address
   from unmatched_addresses as a, zipdata as b
   where calculated COMPGED_score lt 600 and
   (a.street_suffix_final=b.street_suffix
   and a.zip5=b.zip_code
   and a.even_odd=b.primary_even_odd_both
   and a.street_number >= b.primary_low_address
   and a.street_number <= b.primary_high_address
   )
   ;
quit;

data zipmerge;
   set zipmerge;
   SPEDIS_score=SPEDIS(street_name,street_name_zip);
   num1=prxchange("s/[^0-9]//",-1,street_name); lennum1=length(num1);
   num2=prxchange("s/[^0-9]//",-1,street_name_zip); lennum2=length(num2);
   if lennum1=lennum2 then do;
      if num2='' then nummatch='Y';
      else if num1=num2 then nummatch='Y';
          else nummatch='N';
   end;
   else if lennum1 > lennum2 then do;
      if num2=substr(num1,1,lennum2) then nummatch='Y';
          else nummatch='N';
   end;
   else nummatch='N';
   street_name_orig=street_name;
   street_name=street_name_zip;
run;

proc sort data=zipmerge out=zipmerge2;
   by &key descending nummatch COMPGED_score SPEDIS_score;
run;
```

Exhibit 1 continued

```
data zipmerge2;
   set zipmerge2;
   by &key;
   if first.&key;
   ** RC 0515 Only keep matches within certain COMPGED_score and SPEIDS_score parameters **;
   if 10 <= COMPGED_score <= 100 and SPEDIS_score > 100 then delete;
   if 110 <= COMPGED_score <= 190 and SPEDIS_score >= 50 then delete;
   if 200 <= COMPGED_score <= 290 and SPEDIS_score > 30 then delete;
   if 300 <= COMPGED_score <= 390 and SPEDIS_score >= 30 then delete;
   if 400 <= COMPGED_score <= 590 and SPEDIS_score > 25 then delete;
run;

proc sort data=unmatched_addresses;
   by &key;
run;

data unmatched_addresses;
   merge unmatched_addresses (in=int) zipmerge2 (in=inb);
   by &key;
   drop zip4_add_on_high zip4_add_on_low street_suffix_zip4 street_predirection
        street_postdirection secondary_address_type secondary_even_odd_both
               secondary_high_address secondary_low_address street_name_orig street_name_zip
        COMPGED_score SPEDIS_score nummatch num1 num2 lennum1 lennum2;
   if int and not inb;
run;

data vendata.&vendor._nomatch_&tagdate;
   set vendata.&vendor._nomatch_&tagdate unmatched_addresses;
   if &key ne ' ';
run;
```

Exhibit 2

Too Fuzzy - Accepting All Matches



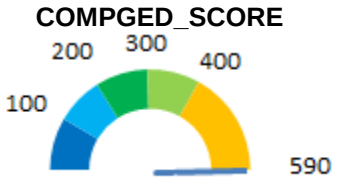
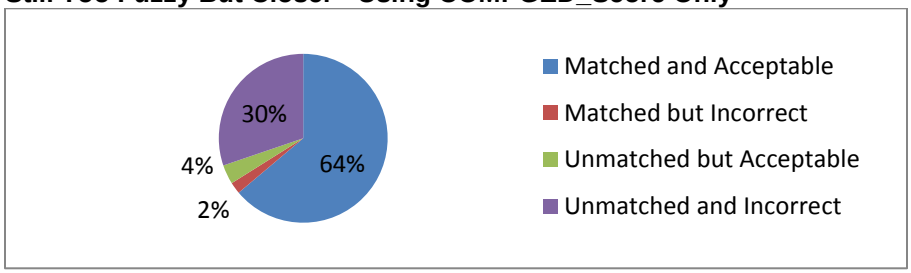


Exhibit 3

Still Too Fuzzy But Closer - Using COMPGED_Score Only



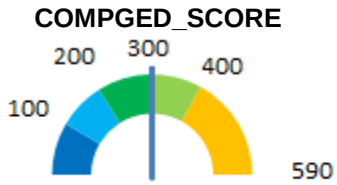
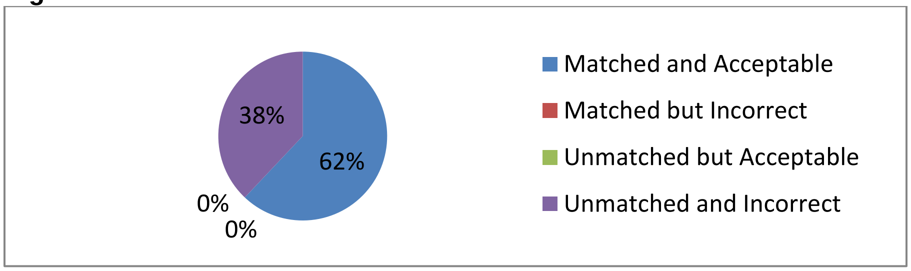
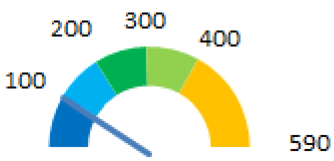


Exhibit 4

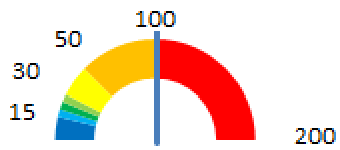
Tightest – Strict Match With No Errors



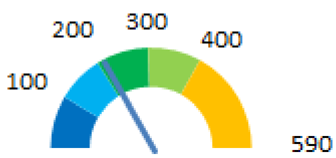
COMPGED_SCORE



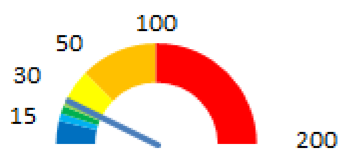
SPEDIS_SCORE



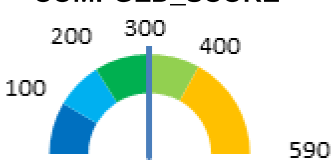
COMPGED_SCORE



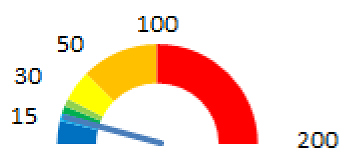
SPEDIS_SCORE



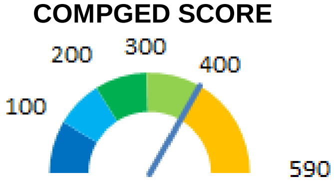
COMPGED_SCORE



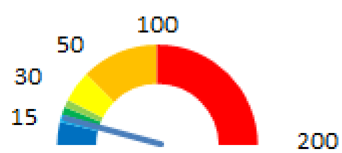
SPEDIS_SCORE



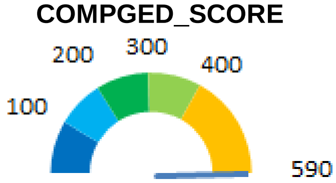
COMPGED SCORE



SPEDIS_SCORE



COMPGED_SCORE



SPEDIS_SCORE

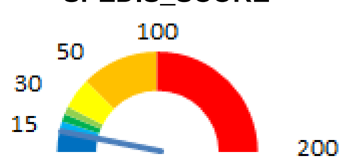
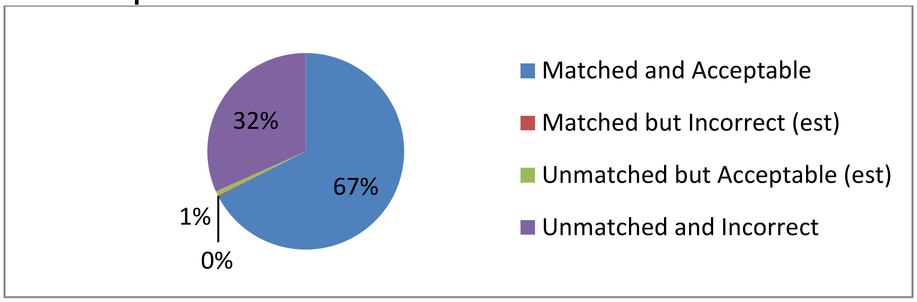
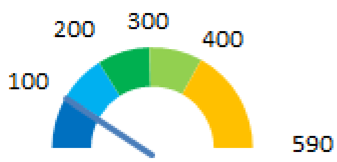


Exhibit 5

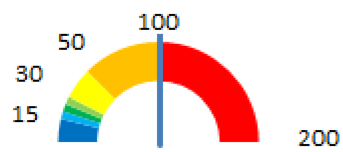
Final Compromise – 5 Thresholds



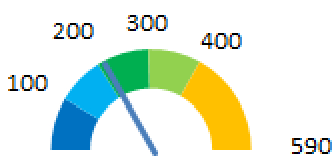
COMPGED_SCORE



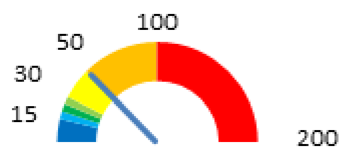
SPEDIS_SCORE



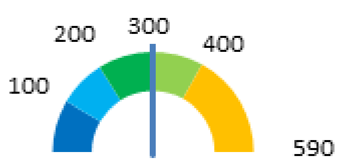
COMPGED_SCORE



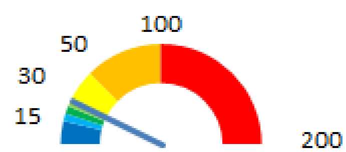
SPEDIS_SCORE



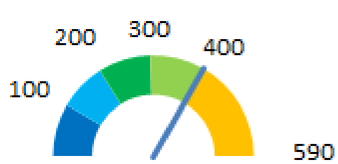
COMPGED_SCORE



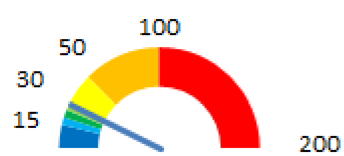
SPEDIS_SCORE



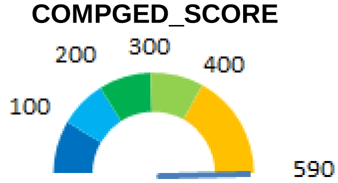
COMPGED_SCORE



SPEDIS_SCORE



COMPGED_SCORE



SPEDIS_SCORE