

# Fuzzy Name-Matching Applications

Alan Dunham, Greybeard Solutions, LLC

## ABSTRACT

Fuzzy matching functions available in SAS allow efficient searches for similar character strings such as names, identification numbers, and addresses. This includes SPEDIS, COMPLEV, COMPGED, SOUNDEX, and the sounds-like operator (=\*). Using those functions is challenging, because even if the data fields to be matched have been scrubbed through a thorough ETL (extract, transform, and load) process, there will be hurdles caused by inconsistent humans, such as spelling errors, typographical errors, entering only some parts of a name, swapping surname and given name, and entering titles as part of the name in one dataset but not in another. This paper provides applied SAS macro code which focuses on the selectable accuracy of matches between two lists of names, regardless of name length, mis-matched titles, diacritical marks, and mis-matched order of a surname and any number of given names.

## INTRODUCTION

The fuzzy-matching code described in this paper has been employed for using a list of 'query' names to search a list of 'target' names for matches. The words target and query are used for convenience, and could refer, for example, to a list of students taking a college course (the query list) with a list of those students who turned in a graded paper (the target list), and the matching operations could be used to determine who turned in a paper. The output of a SAS run would be the records of satisfactory matches (meets score criterion) or, alternatively, could as easily be those which did not achieve a satisfactory match (minor modification of code in this paper). The code in this paper outputs records which achieve satisfactory matches. The code could be modified readily to match addresses or identification numbers.

There are many sources of name-matching challenges other than misspellings and data entry errors. Suppose our example query dataset list had a name field that showed "surname, given name middle name" (e.g., Dunham, Alan Douglas) but the college paper-submittal system swapped the names and left out the middle name (e.g., Alan Dunham) or, worse, A Dunham. Perhaps the student is a doctor and the honorific "Dr." is part of the first name in one dataset list but not the other. In some datasets, apostrophes or diacritical marks such as tildes are permitted. In names of Anglo-Saxon origin, it is common to have three names, but often the middle name is not entered into an automated system. Alternatively, there might be no middle name and the person has to constantly type NMI into automated systems for "no middle initial". Some ethnicities commonly have four or more names and individuals may variously enter two, three, four, or more of their given names, not necessarily in the same order. Some ethnic cultures commonly provide their surname first and other cultures provide their surname last. Many languages have non-Roman alphabets, such as Hebrew, Cyrillic, Arabic, Korean, Japanese, and Chinese, and there can be multiple methods of converting those character sets into English (so-called Romanization). Of course, identification numbers help to alleviate many of those challenges, but often are not available and still have sources of error.

The code in this paper sets up the name fields for matching through SAS function calls and then tests for a match in two ways. First, all vowels, blanks, titles and diacritical marks are deleted from the two text strings for comparison. Second, the individual surname and given names in the query name field all are tested for a match individually against all of the individual surnames and given names in each target name field. Matching without vowels helps with alternative methods of Romanization and spelling errors. The second type of exhaustive matching of each subname in any order does not appear to be a common technique.

Each matching operation generates a score, which is compared to a criterion maximum score. If any of the two matching scores for a given comparison is less than the criterion, the record is output.

In SAS function calls, SPEDIS and COMPGED can be used to compute a score for the degree of matching between two character strings by assigning penalty scores for each operation needed to achieve a match, including substitution, insertion, transposition, and truncation. COMPGED is designed to be more efficient (faster) than SPEDIS. COMPLEV counts the smallest number of insert, delete, or replace operations needed to make the strings match, and thus is a streamlined (faster) but less versatile version of the COMPGED function. SOUNDEX and its interchangeable cousin, the sounds-like operator, work well in the National Archives for looking up the names of American ancestors in Census documents. However, they would not be expected to do well at matching non-English names, and in any case generate too many matches to be used alone for matching large sets of character strings.

The code shown in this paper uses the SPEDIS function to generate the matching score because the customer version of SAS available was SAS 8.02. For later versions of SAS, COMPGED could be substituted for SPEDIS in the code with minor effort. There are other limitations in SAS 8.02 which are easier in later versions, such as counting the number of subnames.

There are some previous papers listed in References below which describe the SAS fuzzy matching functions in great detail, but this paper is focused on usable code that employs the matching function and other string manipulation functions. A good starting point for understanding how the matching functions work is the Staum paper.

## FUZZY NAME-MATCHING CODE

The following code is interspersed with explanatory text in this font, plus there are many SAS code comments in blue.

Set up the macro variables for later use. This facilitates use of user dataset names and tuning the matching score criteria.

```
%let Matches = Test.Matches          ***Desired output dataset of matches;
%let QueryNames = Test.Query_Names;   ***Dataset with query names to match;
%let TargetNames = Test.Target_Names_ ***Dataset to search for matching names;
%let Query=Desired_Name;              ***Actual name of the query name column;
%let Target = Candidate_Name;         ***Actual name of the target name column;
%let Max_Score = 50;                  ***Cutoff for accepting noblanks/novowels matches...
                                     Change to tune for your needs...lower is more strict;
%let Max_Subname_Score = 50;         ***Cutoff for accepting subname matches;
```

All work is done within a data procedure. Several temporary variables are used and dropped from the output.

```
Option obs = max;
Data &Matches (drop= target_subname query_subname i j k m initial_subscore
    Used_score divisor query_subname_count target_subname_count goodcount used
    initial_subscore used_score positionMR positiondash count);
```

Two main loops work through all of the records in the query and target datasets, using the 'querynobs' counter and the 'obsnum' counter. Note the use of the 'point' capability, which allows access to specific records.

```
Do queryNobs = 1 to lastquery;
Set &QueryNames point=queryNobs nobs=lastquery;
    *** The point= option allows control of which observation is read;
Length P $40
    Target_subname      $40
    Query_name          $40
    Query_subname       $40
    Overall_sub_score   3
    Initial_subscore    3;
Array subname_scores(10,10) _temporary_ ;
    ***Array to store match score for each subname combination;
Array subname_minimum_scores (10) _temporary_;
    ***Array to store smallest match score for each subname combination;
Array target_subname_used(10) _temporary_;
    ***track when target subname is used in match to query subname;
Format overall_sub_score 5.1;
Initial_Subscore = 70;
    ***Set subname score to non-match at start of each matching loop;

    ***The following loop goes through every record in the target dataset for
    each name in the query dataset;
do obsnum = 1 to lastrec;
    set &TargetNames point=obsnum nobs = lastrec;
    ***Set to all uppercase and remove diacritical marks;
    Query_Name = upcase (compress(&Query,"()'.,"));
    Target_Name = upcase (compress(&Target,"()'.,"));
```

The technique for deleting titles from the names is shown using 'MR', and the 'dash' is shown being deleted. Other common titles and punctuation, which generally do not contribute to name matching, are not shown being deleted to conserve the length of this paper.

```
***Replace titles and Dashes with a blank;
***Note: only the title "MR" is shown here to display the methodology;
***Note: this technique can be more efficient in versions after SAS 8.02;
```

```

positionMR = indexw(Query_Name,"MR");
positiondash = indexw(Query_Name,"-");
if positionMR > 0 then substr(Query_Name, positionMR,3)="";
if positiondash > 0 then substr(Query_Name, positiondash,1)="";
positionMR = indexw(Target_Name,"MR");
positiondash = indexw(Target_Name,"-");
if positionMR > 0 then substr(Target_Name, positionMR,3)="";
if positiondash > 0 then substr(Target_Name, positiondash,1)="";

```

Names frequently have excess blanks which are deleted to prevent them from affecting name matching.

```

***Get rid of excess blanks;
Query_Name = compbl(left(Query_Name));
Target_Name = compbl(left(Target_Name));

```

The author has found through practice that there is wide variation in the application of vowels in Romanization schemes, and thus simply deleting all vowels helps make the matching run more efficient with little loss in matching results. The resulting names used for the first type of matching are crunched-up strings of left-justified consonants. If readers want to include vowels for matching, the following code could readily be modified and added to delete only spaces for an additional matching operation for each name pair.

```

*** Transform names to have no vowels and no blanks;
Query_Name_Novowel = compress(left(Query_Name), 'AEIOUY ');
Target_Name_Novowel = compress(left(Target_Name), 'AEIOUY ');
*** compute matching scores for names with no vowels and no blanks;
NoVowel_Name_Score = spedis(Query_Name_Novowel, Target_Name_Novowel);

```

The following code works with the multiple subnames in the query and target names to find the best matches regardless of the order in which the subnames occur. This search for matches will find needles in haystacks of names. If there are no matches output based on this subname matching, then it is more likely that no human would find matches through manual searches by subject matter experts.

```

Count = 0;
***pre-set subname score arrays to initial values;
Do k = 1 to 10;
    Subname_minimum_scores(k) = initial_subscore;
    Target_Subname_Used(k) = 0;
    Do m = 1 to 10;
        Subname_scores(k,m) = initial_subscore;
    End;
End;
Overall_sub_score = 0;
*** Determine the number of subnames in each name string;
Query_subname_count = length(compbl(left(Query_Name))) -
    Length(compress(Query_Name)) + 1;
Target_subname_count = length(compbl(left(Target_Name))) -
    Length(compress(Target_Name)) + 1;

```

Each query subname is tested against each target subname, with results stored in temporary arrays to find the minimum matching score for each query sub-name. The minimum scores are added to obtain an overall subname matching score. Note the use of the 'scan' function in conjunction with subname counters and detection of blank spaces to select each subname or comparison.

```

*** Loop through both sets of subnames to obtain match scores;
Do i = 1 to Query_subname_count;
    Do j = 1 to target_subname_count;
        Target_subname = scan(P, j, ' ');
        Query_Subname = scan(Query_name, i, ' ');
        Subname_scores(i,j)=spedis(query_subname,target_subname);
    End;
End;
*** Loop through subnames to find lowest match scores;

```

```

Do k = 1 to query_subname_count;
  Do m = 1 to target_subname_count;
    If (target_subname_used(m) < 1)
      And (subname_scores(k,m) <=
        subname_minimum_scores(k))
    Then do;
      Subname(minimum_scores(k) =
        = subname_scores(k,m);
      Used_score = m;
    End;
  End;
End;
***keep track of when a subname is used for a minimum score;
If used_score = . then used_score = 1;
Target_subname_used(used_score) = 1;
End;
*** Loop through subnames to add up smallest subname scores;
Do k = 1 to query_subname_count;
  Overall_sub_score = overall_sub_score +
    Subname_minimum_scores(k);
End;

```

When the query name and target names have unequal numbers of subnames, not uncommon, a correction is made with a simple index based on the number of subnames in each. This is not proven science, but rather is a practical approach. Readers can create and test their own approach.

```

Divisor = query_subname_count;
*** Correction when Query name has fewer subnames than target name;
If query_subname_count < target_subname_count then do;
  Divisor = target_subname_count;
  Overall_sub_score = overall_subscore +
    (overall_subscore / query_subname_count)*
    (target_subname_count-query_subname_count);
End;
Overall_sub_score = int(overall_sub_score/divisor);

```

If two subnames in the query name have acceptable matches in the target name, a flag is set to output the record. This helps guard against the situation where a person used, say, two of his or her names in one input dataset entry process, and three names in the other dataset entry, so that the lack of match for a third name does not prevent detection of an acceptable match.

```

***Determine if at least two sets of subnames have an acceptable match;
Two_name_flag = 0;
Goodcount=0;
Do k=1 to query_subname_count until(two_name_flag);
  Used = 0;
  Do m = 1 to target_subname_count until(used);
    If(subname_scores_k,m) <= &MaxSubname_Score)
      Then do;
        Goodcount + 1;
        Used = 1;
      End;
  End;
  If(goodcount >= 2 then two_name_flag = 1;
End;
End;
***Output this record if there is some sort of acceptable match;

If overall_sub_score <= &max_subname_score
  Or two_name_flag >= 1
  Or Novowel_name_score <= &max_score
  Then output;

```

```
End;
```

```
End;
Stop;
Run;
```

## CONCLUSIONS

The code shown has been used extensively with some success running SAS 8.02 on a PC. However, the cost-benefit of the exhaustive form of subname matching shown above for a given problem would depend on a variety of factors, including the number of names in each list, the amount of memory available to the SAS run, and how important it is to find matches. The author perceives that the above could be more efficient in later versions of SAS, but has no benchmarks to suggest the degree of improvement.

Both the SPEDIS and COMPGED functions have more sophisticated capabilities which were not exploited in this code. The reader is encouraged to start with the Staum paper in References below to explore improvements.

What is not shown above, of course, is what is done next with the list of candidate matches. There may be further automated matching using other factors such as address or phone number. It may be that completely automated procedures are sufficient for a given matching problem, and no humans need to intervene. For some problems, at some point a human may have to review a candidate list to make a final determination. Humans are superb at matching, especially as subject matter experts, but quickly become overwhelmed if the candidate list of matches is long. That is a problem-specific context that each user will have to explore.

## REFERENCES

An essential reference and starting point for exploring fuzzy matching is:

Staum, Paulette . "Fuzzy Matching using the COMPGED Function," Paul Waldron Consulting, West Nyack, NY, available at <http://www.lexjansen.com/nesug/nesug07/ap/ap23.pdf> .

Other references:

Roesch, Amanda. "Matching Data Using Sounds-Like Operators and SAS® Compare Functions," Educational Testing Service, Princeton, NJ available at <http://support.sas.com/resources/papers/proceedings12/122-2012.pdf> .

Schreier, Howard. "Using Edit-Distance Functions to Identify "Similar" E-Mail Addresses," U.S. Dept. of Commerce, Washington, DC, available at <http://www2.sas.com/proceedings/sugi29/073-29.pdf> .

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Alan Dunham  
Greybeard Solutions, LLC™  
PO Box 10841  
Burke, Virginia 22009  
[alan@greybeard-solutions.com](mailto:alan@greybeard-solutions.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.