Record Linkage Software

User Documentation

Submitted by

System Resources Corporation
400 Virginia Avenue, SW, Suite 350
Washington, D.C. 20024

and

Synetics for Management Decisions, Inc.
3030 Clarendon Boulevard, Suite 305
Arlington, Virginia 22201

U.S. BUREAU OF THE CENSUS

TABLE OF CONTENTS

1.0 BACKGROUND ......................................................... 1
1.1 PURPOSES OF RECORD LINKAGE ........................................ 1
1.2 THEORY AND METHODOLOGY OF RECORD LINKAGE ..................... 1
2.0 RECORD LINKAGE COMPUTER PROGRAMS ............................. 3
2.1 CENSUS BUREAU PROGRAMS ........................................... 3
2.2 OVERVIEW OF THE SYSTEM .......................................... 4
2.2.1 General Overview ............................................... 4
2.2.2 Data Standardization ........................................... 4
2.2.3 Sorting and Blocking ........................................... 6
2.2.4 Record Matching and Clerical Review ............................. 6
2.2.5 Iteration ..................................................... 6
3.0 DATA PREPARATION FOR RECORD LINKAGE ......................... 7
3.1 DATA PREPARATION REQUIREMENTS .................................. 7
3.2 DATA STANDARDIZATION PROGRAMS .................................. 8
3.3 RECORD BLOCKING .................................................. 10
4.0 NAME AND ADDRESS STANDARDIZATION ............................. 13
4.1 STANDARDIZATION OPTIONS ......................................... 13
4.2 USE OF GDRIVER FOR NAME AND ADDRESS STANDARDIZATION ....... 13
4.3 USE OF GDRIVER FOR NAME STANDARDIZATION ONLY ............... 20
4.4 USE OF GDRIVER FOR ADDRESS STANDARDIZATION ONLY .......... 21
4.5 NAME STANDARDIZATION REFERENCE FILES USED IN GDRIVER ....... 21
4.6 ADDRESS STANDARDIZATION REFERENCE FILES USED IN GDRIVER .... 23
5.0 SORTING AND BLOCKING RECORDS FOR MATCHING ................ 25
6.0 RECORD MATCHING PROGRAM .......................................... 27
6.1 SPECIFYING THE LOCATION OF THE INPUT FILES .................... 27
6.2 CREATING THE PARAMETER FILE .................................... 29
6.3 RUNNING THE PROGRAM ............................................ 39
6.4 INTERPRETING THE OUTPUT ......................................... 40
6.5 USING THE RESIDUALS ............................................. 41
6.6 EXECUTING THE SECOND AND SUBSEQUENT MATCHING RUNS .......... 43
7.0 ADVANCED AUTOMATIC ADJUSTMENT FEATURES ..................... 45
7.1 CREATING SORTED DATA FILES ...................................... 45
7.2 CREATING THE PARAMETER FILE PARMS.TXT ......................... 46
7.3 RUNNING THE COUNTER PROGRAM ..................................... 49
7.4 CREATING PARAMETER FILE PARMS.TXT ............................. 49
7.5 RUNNING THE U-PROBABILITY PARAMETER GENERATION PROGRAM .... 53
7.6 RUNNING THE EXPECTATION-MAXIMIZATION (EM) PROGRAM ............ 53
APPENDIX: RUNNING A MATCH ON A SAMPLE FILE WITH TWO PASSES .... 55
A.1 GENERAL DESCRIPTION OF SAMPLE MATCHING JOB .................. 55
A.2 SETTING UP THE MATCHING RUN ..................................... 56
A.3 RUNNING THE MATCHER .............................................. 56
A.4 REVIEWING THE OUTPUT AND CHOOSING CUTOFFS ..................... 57
A.5 RUNNING THE RESIDUALS ............................................ 58
A.6 SETTING UP THE SECOND MATCHING RUN ............................. 59
A.7 RUNNING THE MATCHER—SECOND PASS ................................ 60
LIST OF FIGURES

1.0 BACKGROUND

This section provides a background and context for the software by presenting the purposes of record linkage and the statistical theory and methodology behind it.

1.1 Purposes of Record Linkage

Record linkage is an effort to identify records on two different electronic files that contain information about the same “entity.” These records often refer to a person, organization, or institution. However, the entity could also be a dwelling unit, a real estate property, a criminal case, a license to carry out an activity such as the sale of pesticides or drugs, or anything else that could be the subject of an electronic record.

There are two basic reasons to try to link records: data collation and list construction.

- **Data Collation**

  A project might require data that are not all available from the same source. Such a project might involve checking the consistency between earnings reported on income tax returns and earnings reported by employers to the Social Security Administration. A research study might require comparing data from the Census of Agriculture with data from the Economic Censuses.

- **List Construction**

  Some projects require a list of all the members of a population to serve as a sampling frame, the contact list for a census, or for collation with data from other sources. It often happens that there is no single list of the population, but that a combination of several lists can be expected to include all or most of it. Usually, there is some overlap between these lists. To avoid biasing the sample or census, we must delete duplications so that each member of the population is included only once. This “unduplication” requires that we identify people, or other entities, that are included in more than one list.

1.2 Theory and Methodology of Record Linkage

The concepts of record linkage were introduced by Howard Newcombe in the late 1950’s. The programs documented here are based on statistical theories first developed by two Canadian statisticians, Fellegi and Sunter.

There are two kinds of record linkage: one-to-one matching and many-to-one matching.

- **One-to-one**

  One-to-one matching is used if each entity is represented once on a particular file. The record linkage seeks to establish unique pairs of records, both referring to the same entity.
Many-to-one matching links several records in one file to a single record in another file. For example, imagine wanting to link a file of export licenses and a file of businesses. Each company or establishment in the business file might hold several export licenses. In this case, the purpose of the record linkage would be to match the appropriate business to each of the export licenses.

Many-to-one matching: An example

One common example of many-to-one matching is "geocoding," so called because one of its uses is to match addresses to larger geographic units such as ZIP codes or Census blocks or tracts.

2.0 RECORD LINKAGE COMPUTER PROGRAMS

Computer programs have been developed to perform record linkage tasks. The programs developed by the Bureau of the Census and documented here are designed to perform one-to-one matching.

Outside programs such as the commercially-available AutoMatch must be used to perform many-to-one matching. AutoMatch is available within the Bureau on a few selected computers; the only one accessible to all Bureau personnel is in the Statistical Research Division (SRD).

2.1 Census Bureau Programs

Census has developed four different programs to perform different parts of the record linkage process.

- **SRD Name Standardizer**
  The Statistical Research Division developed a program to standardize the names of businesses and individuals.

- **GD Address Standardizer**
  The Geography Division developed a program to put business and personal addresses into a common format.

- **GDREVR Program**
  The Statistical Research Division GDREVR program integrates the name and address standardizers and allows both names and addresses to be converted to a common format in a single computer run. This program can also be used as a driver for the name standardizer alone or the address standardizer alone. These programs could be used alone if the files do not include both names and addresses or if another standardization program is used for one of the elements.

- **SRD Record Linkage Program**
  The Statistical Research Division also developed a record linkage program. Since standardization of information is a key element of the linkage process, this program depends on use of the standardizer programs or other programs that can standardize names and addresses.

- **DMD Name Standardizer**
  The Decennial Management Division also developed a Name Standardizer that puts the names of individuals into a common format. This program is not documented here. The available documentation indicates that it is particularly appropriate for standardizing files that include military personnel. For more information about this program, call Anne McGaughey, now (March, 1999) in the Decennial Statistical Studies Division, at 301-457-4110.
2.2 Overview of the System

This section includes a general overview of the process of record linkage as it is performed by the Bureau of the Census programs.

2.2.1 General Overview

Figure 1 is a general flowchart of the record standardization and matching process. That process begins with two electronic files, designated File A and File B. (These file designations are used in the statistical theory of record matching and will be used throughout this documentation.)

1. Both files must first be standardized using the programs described in Section 4.

2. The standardized files must then be sorted according to the blocking variables, producing a pair of sorted files.

3. It is these sorted files that the matching program can collate.

4. After each pass through the matching program, record pairs are assigned either to a file of matched records, or to a "residual" file of unmatched records.

These unmatched records must be re-processed by a new set of blocking variables. The re-processed files can then be run through the matching program just like the original two files.

The process of examining unmatched records, re-sorting, and rematching can be repeated until no or almost no new matches are produced.

At that point, the user has a set of files of matched records to use in his project and the functions of the matching software have been completed.

2.2.2 Data Standardization

Before any attempt is made to link records, the data in them must be standardized.

The Bureau of the Census record matching system includes modules to standardize names, both business and personal, and street addresses. A single driver program, GDRIVER, allows the user to standardize both names and addresses within a single file. The names and addresses in the second file must be standardized in a separate step.
2.2.3 Sorting and Blocking

The record matching program will work only on files that have been sorted and standardized. These standardized files must be sorted in the order of the blocking variables.

The Census system makes use of information provided by the user for blocking, but does not include a utility for sorting the records into the required order.

- Section 5.0 of this documentation includes instructions for using a freeware sorting program.
- If large files are being matched, the user might have to use a different sorting program with a larger capacity, such as the sort utility of SAS.

2.2.4 Record Matching and Clerical Review

Once the data have been standardized and sorted, the matching module can be executed.

The output of the matching program includes a sorted list of pairs of records with their matching scores.

The user must examine this list by hand ("clerical review") to determine the minimum score above which all matches found by the program are, in fact, matched pairs, and the score below which no pairs are true matches. Usually, there will be a range between these two numbers that includes some pairs that really are matches and some that the user can determine are not or are probably not true matches. Use of the output files of matches is not further supported by the matching program.

2.2.5 Iteration

The matching program also identifies "residuals," records that are not involved in matched pairs.

It is usually advisable to run the residual records through the matching program again after adjusting the matching score parameters and sorting and blocking on a somewhat different set of variables. A second matching run should produce a different set of matched pairs of records. It is up to the user to merge these records with the matches produced on the previous matching run.

This iterative process may produce additional matches on a second pass.

---

3.0 DATA PREPARATION FOR RECORD LINKAGE

Record linkage programs must either assume that the data to be matched are in a common format, leaving the user to ensure that this is the case, or include a standardization module or subroutine.

3.1 Data Preparation Requirements

Record linkage often involves electronic files compiled at different times or by different organizations, using different conventions for formatting and abbreviating the data. This causes differences in the format of identical information.

(1) Even when data are collected by the same organization, different individuals might respond to different information requests and thus use different name and address formats.

(2) The same individual might respond in different ways at different times, such as by omitting a middle name or initial on one form but including it on another, or abbreviating part of his address on one but not on another.

For example, compare the following two business records:

- Northwest Regional Hospital
  Department of Obstetrics & Gynecology
  223 Franklin D. Roosevelt Drive

- OB/GYN DEPT
  NW REG HOSP
  223 FDR DR

Similarly, compare the following two personal records:

- Dr. William S.T. Frelinghuysen III
  501 East 10th Street, Apartment 8-D
  New York, New York 10011

- WM S. FRELINGHUYSEN
  501 E 10 ST #8D
  NY, NY 10011-2345

Although each pair of records pertains to the same hospital or individual, within each pair the two records differ in the information included:

(1) One record includes Dr. Frelinghuysen's title, a generational suffix, and two middle initials. The other record omits the title, suffix, and one of the middle initials.

(2) One record is in upper and lower case letters. The other is all in capital letters.

(3) One record makes the maximum use of abbreviations. The other uses no abbreviations.

A computerized linkage program is not likely to connect these pairs of records unless the information is formatted much more consistently.
3.2 Data Standardization Programs

The data standardization programs documented here work by "parsing" name or address elements, that is, breaking them down into a set of elements that are then rearranged in a common order and abbreviated in common ways.

For instance, the SRD name standardizer recognizes eight elements (coded 1 to 4 and 6 to 9) that might be found in a personal name:

<table>
<thead>
<tr>
<th>Code</th>
<th>Element</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Conjunction</td>
<td>and, &amp; or</td>
</tr>
<tr>
<td>2</td>
<td>Generational suffix</td>
<td>Senior, Jr., III, 3rd</td>
</tr>
<tr>
<td>3</td>
<td>Title</td>
<td>Professor, Dr., Ms.</td>
</tr>
<tr>
<td>4</td>
<td>Occupational suffix</td>
<td>M.D., Ph.D., C.P.A.</td>
</tr>
<tr>
<td>5</td>
<td>Initial</td>
<td>A., B.</td>
</tr>
<tr>
<td>6</td>
<td>Last name</td>
<td>Smith, Martinez</td>
</tr>
<tr>
<td>7</td>
<td>Last name prefix</td>
<td>Saint, da, los, van, Mac</td>
</tr>
<tr>
<td>8</td>
<td>First name</td>
<td>James, Mary, Rafael</td>
</tr>
</tbody>
</table>

The program compares these elements to lists in files that it accesses:

- The list of conjunctions is quite short, the list of first names is quite long.
- A user dealing with a special population can improve the standardization of the data by adding terms to one of these lists.
- The Census name standardizer recognizes the various elements in whatever order they occur, converts variations like nicknames to a standard form, and rearranges elements in a common order before consolidating them into three fields: first name, middle name, and last name.

A similar process can be performed with address elements. The Bureau of the Census address standardizer recognizes 10 elements that might be found in a street address:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>House number</td>
<td>123</td>
</tr>
<tr>
<td>Street name prefix</td>
<td>N, S, E, W</td>
</tr>
<tr>
<td>Main street name</td>
<td>Oak, Main, Martin Luther King</td>
</tr>
<tr>
<td>Secondary street name</td>
<td>Avenue, Blvd., Place, Court</td>
</tr>
<tr>
<td>(street type)</td>
<td></td>
</tr>
<tr>
<td>Within-structure description</td>
<td>Bldg., Apt. #</td>
</tr>
<tr>
<td>Within-structure identifier</td>
<td>S-W, 405</td>
</tr>
<tr>
<td>Rural route description</td>
<td>RR</td>
</tr>
<tr>
<td>Rural route identifier</td>
<td>Usually a number</td>
</tr>
<tr>
<td>Box description</td>
<td>Box, PO, PO Box</td>
</tr>
<tr>
<td>Box identifier</td>
<td>A letter or number</td>
</tr>
</tbody>
</table>

The address standardization program refers to a file of possible patterns to recognize these elements by their content and position, and rearranges them into a standard order. At the same time, the program converts elements like street types to standard abbreviations.

Both the name and address standardization programs refer to files of name and address elements that are based on experience gained in previous matching projects. These files are described in Sections 4.5 and 4.6.

- A user may want to modify these files to reflect the particular populations or type of records being examined.
- You must be familiar with the data in the files you are going to be matching. It may be necessary to do one or more matching runs and review a sample of the matches to determine what kind of terms need to be added.

If you think the matching program is not recognizing correct matches, you may want to consult with the Statistical Research Division about adapting the program to your data.
3.3 Record Blocking

The main objective of a record linkage program is to classify every possible pair of records as either a matched pair or an unmatched pair. It would be impractical to compare all record pairs between two files of any reasonable size, since the total number of pairs is the product of the number of records on each file. Therefore, the records on each of the files being compared must be blocked.

Blocking is a division of an entire file into mutually exclusive subsets.

When records on both files are blocked, the records on File A are compared only to records in the corresponding block on File B.

If there were still too many comparisons to execute within a reasonable time after a first selection of blocking variables, both files would have to be divided more finely.

Since blocking precludes most of the possible matches from being checked, selection of the blocking variables is a very important step in the record linkage process. For instance, if two files both contain addresses, they can be blocked by state. This would divide each file into 50 blocks and greatly reduce the number of potential matches to be attempted.

Comparing files for duplication without blocking: An example

Determining whether there are any duplications in two small files of 1,000 cases each would require 1,000 x 1,000 = 1,000,000 comparisons. In this example, at most 1,000 matches could be found, and the remaining 999,000 would be "unmatched" pairs. Determining whether there are any duplications in two modest-sized files of 10,000 cases each would require 10,000 x 10,000 = 100,000,000 comparisons. Determining whether any of the 10,000 cases in a modest-sized file are already included in a large file of 20 million records would require 10,000 x 20,000,000 = 200,000,000,000 or 200 billion comparisons. This is too many for the Bureau of the Census computers to handle within a reasonable run time.

Comparing files for duplication with blocking: An example

If two 10,000-record files were each divided into 20 equal blocks of 500, instead of making one set of 100 million comparisons, the linkage program could make 20 sets of 500 x 500 = 250,000 comparisons for a total of only 250,000 x 20 = 5,000,000 comparisons. Note, however, that dividing a 20 million-record file into 20 blocks would still leave 1 million records per block, and even against blocks of only 500 in the smaller file, within each pair of blocks there would still be 500 million comparisons. Performing 500 million comparisons within 20 blocks would require a total of 10 billion comparisons instead of 200 billion.

Several factors must be considered before deciding to block on state, however.

First, although blocking on state would create 50 blocks (51 if the District of Columbia were included—even more if the files included Puerto Rico and the territories), the blocks would not be of equal size. Although the average state has only 2 percent of the nation's population, California has about 12 percent and Wyoming has only 0.2 percent.

It is necessary, therefore, to ensure that the blocking variables selected do not have any unusually large categories by determining the sizes of the potential blocks in each file and computing their products.

A second problem in blocking involves the accuracy of the data recorded in the blocking variable.

For instance, if through keying errors some Maine addresses have been entered as "MD" (Maryland) instead of "ME," or if some Louisiana addresses have been transposed from "LA" to "AL" (Alabama), then the miscoded addresses will never be compared and the correct matches will not be found.

For example, neither of the following pairs of records, pertaining to the same individual, would be matched:

<table>
<thead>
<tr>
<th>FIRSTNAME</th>
<th>MI</th>
<th>LASTNAME</th>
<th>ST</th>
<th>STREET</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
<th>SEX</th>
<th>DOB</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALFRED</td>
<td>H</td>
<td>SMITH</td>
<td></td>
<td>WALKER</td>
<td>AUGUSTA</td>
<td>MD</td>
<td>04324</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALFRED</td>
<td>H</td>
<td>SMITH</td>
<td></td>
<td>WALKER</td>
<td>AUGUSTA</td>
<td>MD</td>
<td>04324</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FELBERT</td>
<td>Z</td>
<td>ZWACH</td>
<td></td>
<td>IVANHOE PONY</td>
<td>BATON ROUGE</td>
<td>LA</td>
<td>70829</td>
<td>M</td>
<td>027284</td>
<td>9957716635</td>
</tr>
<tr>
<td>FELBERT</td>
<td>Z</td>
<td>ZWACH</td>
<td></td>
<td>IVANHOE PONY</td>
<td>BATON ROUGE</td>
<td>LA</td>
<td>70829</td>
<td>M</td>
<td>027284</td>
<td>9957716635</td>
</tr>
</tbody>
</table>

Each of these two pairs differs only by a single letter in the STATE variable, and any probabilistic matching algorithm would identify each pair of records as pertaining to the same individual. However, if the state code were being used as a blocking variable the pairs of records would never be compared.

It is thus important to block records on variables that are known to be highly accurately coded. In these two cases, the errors in the state codes could be detected by comparing them to the ZIP codes, since Maryland ZIP codes begin with '2,' not '0,' and Alabama ZIP codes begin with '3,' not '7.'

Because of the problems associated with errors in blocking variables, it is important to reblock the unmatched records on different variables and make another attempt to match the records. As noted earlier, the Census matching program outputs the matching records to a separate residual file so that the remaining records can be run through the program again.
4.0 NAME AND ADDRESS STANDARDIZATION

Standardization is a separate process that must be carried out before records can be matched.

Section 4.1 describes the standardization options. Sections 4.2-4.5 specify how to execute each of the options supported by Bureau of the Census software.

4.1 Standardization Options

Data standardization options are shown in Figure 2.

The GDRIVER program is set up to standardize both names and addresses (see Section 4.2), but changing the specifications in one line of a required file—*parmdrs.txt*—will let you standardize names only (see Section 4.3) or addresses only (see Section 4.4).

You can standardize names alone when the files to be matched do not include addresses or the addresses have already been standardized. Likewise, you can standardize addresses alone when the files to be matched do not include names or the names have already been standardized. You may also use any other standardization program to which you have access and which you know how to use.

4.2 Use of GDRIVER for Name and Address Standardization

The following is a step-by-step guide to standardizing both the name and address fields for two files using the GDRIVER standardization program. Perform the following steps on File A and then repeat the steps for File B.

The use of GDRIVER is illustrated in Figure 3. The program calls several reference files, for both name and address standardization. The reference files for name standardization must be in the directory where the standardization will take place. If the reference files for address standardization are not in this directory, the file *initfiles.dat* needs to be created. This file should contain the path of where these files are located. The user must also tell GDRIVER the location of the file of records to be standardized, through the *parnsss.txt* file, and the layout of that file, through *parmdrs.txt*. GDRIVER creates two output files, one of standardized records (*stoutout.txt*) and one of records it could not standardize (*otherout.txt*). It also produces a summary report.

This guide assumes that the two files are fixed-length ASCII files with one record per entity. Words in *italics* are actual filenames.
Figure 2: Standardizer Options

A: Unstandardized Files to Be Matched
B: Standardized Files for Input into Matching Program

Figure 3: GDRIVER Standardizer

GDRIVER.EXE

Name Modules
Address Modules

File of Standardized Names and Addresses
See Figure 4

File of Non-Standard Records

Summary Report

(infile.dat)

Three Lists of Path Names in Ref. Files
See Section 4.6

(blok.dat pref.dat shpl.dat
ser.dat nick.dat three.dat
pss.dat two.dat
ved.dat)

Reference Files for Address Standardization
clr_abbr_4.dat
master_clues.dat
patterns.dat

See Section 4.5

Path Name of Input File

See Section 4.6

(infile.dat)

All files in this frame are in a single directory or "folder."

Parameter.txt

MS NL AS AL KL where:
MS: Main Field Start
NL: Main Field Length
AS: Ada. Field Start
AL: Ada. Field Length
KL: Kernel Length

(Layout of Input File)
4.0 NAME AND ADDRESS STANDARDIZATION

The first step is to prepare the two files that will be standardized.

A. Create a directory for one of the files.

B. Copy the GDPLAYER.EXE file to the directory you just created.

C. Copy the reference files for name standardization to the directory. See Section 4.5 for a full list of and discussion of the reference files for name standardization.

D. Copy the file parnm.txt to the directory.

   The file parnm.txt should contain only the name and path of the file to be standardized.

E. Copy the file parmd.txt to the directory.

   The file parmd.txt should contain a single line specifying the record layout of the file to be standardized in the following format:

   NS NL AS AL RL.

   For example, in parmd.txt:

   NS—Starting column of name field (column 0)
   NL—Length of name field (40 columns)
   AS—Starting column of addressfield (column 41)
   AL—Length of address field (40 columns)
   RL—Total record length (115 columns)

   0 40 41 40 115

F. Copy the file inifiles.dat to the directory.

G. The file inifiles.dat should contain the names and paths of the following three reference files for address standardization (each on a separate line):

   - class_abbrv.dat
   - master_claus.dat
   - patterns.dat

   See Section 4.6 for a full discussion of the reference files for address standardization listed in the file inifiles.dat.

   If these files have been copied to the same directory as the other files discussed here, inifiles.dat will not be necessary.

The next step is to use the GDPLAYER program to standardize the names and addresses in a file. To do this, simply execute GDPLAYER.EXE in the directory. Once the GDPLAYER program has completed its standardization, review the output.

The GDPLAYER program produces four files; each is described here.

- stanout.txt (File of standardized records)—each record of this file contains all of the data from the original records that the program could standardize on both name and address, plus the standardized data. The original record layout remains unchanged. Appended to the end of each record are the standardized address and name fields in the following format:

   Field Name    Length
   House number  6
   Street name prefix direction  5
   Main street name  20
   Secondary street name  10
   Within-structure description  4
   Within-structure identifier  5
   Rural route description  3
   Rural route identifier  5
   Box description  4
   Box identifier  5
   Title  6
   First name  13
   Middle name  13
   Last name  14

The following eight lines are a sample of records from an unstandardized file:

Paula Abdul  771 Teakwood Rd.  Los Angeles CA 90049
Kareem Abdul-Jabbar 1436 Summitridge Dr.  Beverly Hills CA 90210-2246
Brooke Adams  248 E. Von Ness Ave.  Los Angeles CA 90004-3821
Charles F. Adams RR 1 Box 23  Bakersfield CA 93205-1234
Don Adams  2160 Century Park East #110  Los Angeles CA 90067
Backstreet Boys  7380 Sand Lake Rd. #350  Orlando FL 32819
The Bee Gees  4800 San Jacinto Cir. #F  Palmbrook CA 92268-9206
The Bee Gees 20505 US Hwy 15 North #12-290 Clearwater FL 33764

Figure 4 displays five lines extracted from the stanout.txt file. They show what each record looks like after standardization.
- otherrout.txt (File of non-standard records)—this file contains the original records that the program could not standardize on either name or address. The original record layout remains unchanged. Appended to the end of each record are the fields that the program could standardize in the following format:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>House number</td>
<td>6</td>
</tr>
<tr>
<td>Street name prefix direction</td>
<td>5</td>
</tr>
<tr>
<td>Main street name</td>
<td>20</td>
</tr>
<tr>
<td>Secondary street name</td>
<td>10</td>
</tr>
<tr>
<td>Within-structure description</td>
<td>4</td>
</tr>
<tr>
<td>Within-structure identifier</td>
<td>5</td>
</tr>
<tr>
<td>Rural route description</td>
<td>3</td>
</tr>
<tr>
<td>Rural route identifier</td>
<td>5</td>
</tr>
<tr>
<td>Box description</td>
<td>4</td>
</tr>
<tr>
<td>Box identifier</td>
<td>5</td>
</tr>
<tr>
<td>Title</td>
<td>6</td>
</tr>
<tr>
<td>First name</td>
<td>13</td>
</tr>
<tr>
<td>Middle name</td>
<td>13</td>
</tr>
<tr>
<td>Last name</td>
<td>14</td>
</tr>
</tbody>
</table>

Three lines in Figure 4 are extracted from the otherrout.txt file. They show what each record looks like after standardization. Note that these records could only be standardized on some of the fields.

- Sumnap (Summary report)—this report shows the following: 1) the number of entries in each of the reference files; 2) the number of records read; and 3) the number of names standardized. Below is an example of this report.

Conjunction table size is 6
Jr/sr table size is 13
Nickname table size is 491
Pattern table size is 851
Prefix table size is 23
Skip list table size is 1146
Suffix table size is 19
Title table size is 9
Two suffix table size is 5
Three suffix table size is 3
Blank table size is 5

Example of report from sumnap
4.3 Use of GDRIVER for Name Standardization Only

To use the GDRIVER to standardize names alone, follow the same basic procedures described in Section 4.2. The only difference comes in step E. After copying the `paraud.txt` file to the directory, edit this file: enter "0" as the length of the address field. For example:

```
0 30 30 0 100
```

Here the fourth number (0) specifies that the length of the address field is zero, so the GDRIVER program ignores the field. Note that the starting position of the address field (the second 30) isn’t “zeroed” out. The GDRIVER program ignores the starting column when the length is zero.

This change means the `stanout.txt` file would look different than what is shown in Figure 4: the fields for the standardized address are blank. In other words, the output file will show each original input record, 67 blank spaces, and then the standardized name fields in the layout specified in Section 4.2.

---

4.4 Use of GDRIVER for Address Standardization Only

To use the GDRIVER to standardize addresses alone, follow the same basic procedures described in Section 4.2. The only difference comes in step E. After copying the `paraud.txt` file to the directory, edit this file: enter "0" as the length of the name field. For example:

```
0 30 30 0 100
```

Here the second number (0) specifies that the length of the name field is zero, so the GDRIVER program ignores the field. Note that the starting position of the name field (the first 30) remains. The GDRIVER program ignores the starting column when the length is zero.

This change means the `stanout.txt` file would look different than what is shown in Figure 4: the fields for the standardized name are blank. In other words, the output file will show each original input record, 46 blank spaces, and then the standardized address fields in the layout specified in Section 4.2.

---

4.4 Name Standardization Reference Files Used in GDRIVER

There are 11 reference files that must be in the directory where GDRIVER.EXE resides. Each of these files modifies text strings that have been determined through experience to require special attention. These files can be used as is, or you can edit these files, customizing the standardization process to the particular needs of your project. Each of the reference files is listed below, and its corresponding special processing is briefly discussed. The entries in these files must appear in alphabetical order.
RECORD LINKAGE SOFTWARE  4.0 NAME AND ADDRESS STANDARDIZATION

- **blk.dat** A list of words which the standardizer replaces with blanks if they appear in the “name” field; examples include “Construction,” “Excavating,” and “Owner.” (Note: these strings are only blanked when the standardizer associates them with a last name.) These are terms that would appear in the names of businesses but would not be useful for matching because of their common occurrence.

- **conj.dat** A list of conjunctions that may appear in the name field, indicating that two or more standardized records are to be produced; examples include “&,” “+,” and “/&.”

- **jrsr.dat** A list of generational suffixes; examples include “JR.,” “III,” and “2ND.” The software also converts words such as “THIRD” to “III” or “2ND” to “II.”

- **nick.dat** A list of names that are used to replace variations on first or middle names; examples include “Al” replacing “Albert,” “Alber,” “Alvin,” and “Alva” and “Barbara” replacing “Barb” and “Barbie.”

- **pat.dat** A list of name patterns that the different name elements might exhibit; the name standardizer uses these patterns as instructions about how to rearrange the elements from their order in the file being standardized to a common order.

- **pref.dat** A list of prefixes found in last names; examples include “De,” “La,” and “Van-Der.” When the software encounters these prefixes it attaches them to the beginning of the last name. For example, “Jim St John” becomes “Jim St. John.”

- **skpl.dat** A list of words indicating that the entity in question should be skipped by the name standardizer; examples include “Academy,” “Church,” “Theater,” “Inc.”, and “Corporation.” These terms also indicate business or institutional names.

- **suff.dat** A list of occupational suffixes that the standardizer identifies; examples include “D.D.S.,” “Esq.,” and “Ph.D.”

- **three.dat** A list of three-letter occupational suffixes with embedded spaces; the standardizer deletes the spaces and then looks for the suffix in **suff.dat**; examples include “C.F. A” and “D M.D.”

- **title.dat** A list of prefix titles that the standardizer identifies; examples include “Mr.,” “Dr.”, and “Rev.”

---

4.5 Address Standardization Reference Files Used in GDIVER

The file **initfiles.dat** refers to a collection of files that store key address words, their variant spellings, and the various address patterns to be recognized. A brief description of the files is provided below:

- **clue_abbrev.dat** (Clue Word Abbreviation Table)—This file contains all the variant spellings or alternate representations for the key words to be identified. The file also indicates what type or types of keyword this clue could be. For example, “Ave” is a street type; “RD” could be a street type or a rural route type.

- **master_clues.dat** (Master Clue Word Table)—This file has a single record for each keyword. For example, there is one record for “Avenue,” not separate records for “Av.,” “Ave.,” “Avenue,” etc. Each record has a space for several “official” variant forms of the word that might be desired on output, including the full name and an official USPS abbreviation.

- **patterns.dat** (Pattern Table)—The Pattern Table contains input patterns of keyword types, and indicates which output field each of the input key words should get mapped to, as part of the parsing process. Each pattern also has a priority code, which supports a scoring system that is used to resolve ambiguous clue words.
5.0 SORTING AND BLOCKING RECORDS FOR MATCHING

We have presented some of the theoretical considerations concerning blocking records in Section 2.2.3. This section specifies how to execute your blocking strategy. Once you have decided which variables you will block on, you need to sort both of the files that will be used in the match by these variables. The programs distributed by SRD do not include a sort utility. There are many different options available for sorting files, with one such sort utility being QSORT. This is a public domain sorting utility available at the Bureau of the Census, and it is the utility that we are documenting here.

The following is a guide to using QSORT:

(1) Create a directory for the sorting to take place.
(2) Place the QSORT. EXE program in this directory.
(3) Place the file(s) to be sorted in this directory.
(4) Rename any input file having a long file name to one with eight or fewer characters (e.g., STANOUTA.TXT to STANOUTA.TXT).

Once this is done, you are ready to execute the sorting utility. The sort can be run in the Windows or DOS environment. An example of the command needed to execute the sort is as follows:

```shell
QSORT STANOUTA.TXT SORTA.DAT /73:2 /96:5
```

This command is made up of four components:

- **QSORT**: This command calls the program (QSORT.EXE).
- **STANOUTA.TXT**: This is the input file (which will be the output from the standardization program or the unmatched records from one of the matching runs).
- **SORTA.DAT**: This is the name of the file that will be created from the sorting utility.
- **/73:2 /96:5**: This component tells the sort program which variable(s) to sort by. The command must be in this format for each variable; that is, a slash, followed by a starting position, followed by a colon, followed by the number of positions occupied by the variable.
In this example, the sort program will sort first on the data in positions 73–74 and then on the data in positions 96–100.

The output file created from the sort utility will have the same file layout as the input file.

**Example of sorta.dat**

After standardizing File B, rerun the QSORT utility, renaming the input stanouta.txt to stanoub.txt, and replacing sorta.dat with sord.dat as the destination filename.

**Example of sord.dat**

---

**6.0 RECORD MATCHING PROGRAM**

After Files A and B have been standardized and sorted, they can be run through the SRD record matching program. The Appendix describes a sample matching task using two files containing names and addresses.

Figure 5 illustrates the record matching process. After the two standardized files, A and B, have been sorted by the blocking variables, the user must supply the locations of the sorted, standardized files through creation of the file parnm.txt (see Section 6.1). The user must also provide a set of parameters through creation of parmnf.txt (see Section 6.2). The matcher (MATCHER.EXE) refers to these files and produces three output files and a summary report. The three output files are:

- **pnn1.dat** Possible matches
- **pnn2.dat** Unlikely matches with scores less than 0
- **parmnf.dat** A pointer file used in the next step to process and sort the residuals or unmatched records

The possible matches in pnn1.dat must be sorted by descending score; then, run PRINTER.EXE to produce a list of matches from pnn1.dat ordered from highest to lowest score. The user must review this file to decide on cutoff scores to input into the next step. Together, pnn1.dat and pnn2.dat will include all matches with scores between the two cutoff values specified in Set 5 of the parameter file parmnf.txt (see Section 6.2).

**6.1 Specifying the Location of the Input Files**

Create parmn.txt. This file contains two lines with the names and locations of the two input files—File A and File B. If File A and File B are located in the directory where the matching will occur, parmn.txt only needs to contain the file names.
6.2 Creating the Parameter File

Create parmfnt. This file is the user's mechanism for controlling the functioning of the matching program. You may choose to modify many of the third set of variables based on the contents of init.dat (an output from Advance Automatic Adjustment Features).

The parameter file init.dat is optional and is used to replace matching probabilities in parmfnt. For the replacement to succeed, the first three characters of the variable name in init.dat must agree with the first three characters of the corresponding variable name in parmfnt.

An example of parmfnt is shown below. The labels “Set 1,” “Set 2,” etc. are not part of the file but are used for reference. Since some sets vary in number of lines, they are not referred to here by line number.

Please note:
- Each line in the parameter file must end with a blank space.
- Maximum input record size is 499.
- The maximum size of a single field (e.g., first) is 60.

Example of parmfnt

```
Set 1
290100010214229
st 93210370
fei 200121510
last 174121891200.987990.0430
first 174121891200.984000.0778
mds 187122021200.49604
hmm 1025110500.94380.2354
stat 11222012000.665030.3332
unit 14615161500.100.00
box 1633178500.70370.1353
Set 5
city 7117811700.60.4
Set 9
yn 4614146400.80.9
```

Our example does not include Sets 4, 6, 7, and 8 because they are not generally used. However, these sets are briefly described in the appropriate places in the discussion.
RECORD LINKAGE SOFTWARE

6.0 RECORD MATCHING PROGRAM

Set 1 consists of the following parameters. The parameters are separated by spaces.

- Number of blocking fields. (there are 2 blocking fields in the example below, so the program will interpret two lines as containing information about the blocking variables)

2. Number of matching variables. (there are 9 comparison fields in the example, so the program will interpret nine lines as containing information about the matching variables)

3. Number of locations of sequence numbers. Users not familiar with this feature should use 0; 1 is for one location. More than one location may not be specified. (0 is used in the example)

4. Cutoff switch. If this switch is set to 1, the two cutoffs in Set 9 are read in; if it is set to 0, they are not. (the cutoff switch is set to 1 in the example) See Set 9, described below.

5. Number of tables. This must be a value between 0 and 2. Use 0 unless you know how to use the options for 1 and 2 (not documented here). (there are 0 tables in the example)

6. Downweight switch. This switch should always be set to 0 (downweight switch = 0 means that no adjustments are made to the weights). If it is set to 1, 2, or 3 (not documented here), various errors are likely to occur. (the downweight switch is set to 0 in the example)

7. Number of ID fields. This is used for testing in situations where the truth of matches is known (i.e., the user obtains truth data sets, that can be used to improve the matching). When the ID field contains a 1, the pointer file—and the print files—will have the indicator "1" or "2" placed in them. If the number of ID fields is 0, then "0" is placed in all appropriate locations in the pointer file and the print files. (there are 0 ID fields in the example)

8. Print switch. If this switch is set to 1, a subset of the matches will be written to the two files, pruid1.dat and pruid2.dat, from which they may be printed. Together these will include all the matches with scores between the high and low cutoff values specified in Set 9 described below. If the switch is set to 0, no matches will be written to the pruid1.dat and pruid2.dat. (the print switch is set to 1 in the example)

9. Exact switch. This switch activates features in earlier versions of the software that are no longer included; therefore, it should always be set to 0. (the exact switch is set to 0 in the example)

10. Length, in characters, of records in File A. (214 characters long in the example)
Set 2 consists of one line for each blocking field. Each line must include the following information in the order given. The parameters must be separated by spaces.

- Name of blocking field. This parameter must be exactly 20 character positions long. If the name of the blocking field is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the blocking field labels in the example below are st and fc)

- Starting position of the blocking field in File A. ("st" starts in position 93 in File A in the first line of Set 2 in the example)

- Length, in characters, of the blocking field in File A. (the blocking field is 2 characters long in File A in the first line of Set 2 in the example)

- Starting position of the blocking field in File B. ("st" starts in position 103 in File B in the first line of Set 2 in the example)

- Length, in characters, of the blocking field in File B. (the blocking field is 2 characters long in File B in the first line of Set 2 in the example)

- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to 0 in the first line of Set 2 in the example)

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td>93 2 103 2 0</td>
</tr>
<tr>
<td>fc</td>
<td>200 1 215 1 0</td>
</tr>
<tr>
<td>last</td>
<td>200 13 215 13 0 0 0.9799 0.0430</td>
</tr>
<tr>
<td>first</td>
<td>174 12 109 12 0 0 0.9840 0.0076</td>
</tr>
<tr>
<td>mid</td>
<td>187 12 202 12 0 0 0.49 0.04</td>
</tr>
<tr>
<td>hnum</td>
<td>102 5 117 5 0 0 0.8438 0.2354</td>
</tr>
<tr>
<td>stt</td>
<td>112 20 127 20 0 0 0.6630 0.3332</td>
</tr>
<tr>
<td>unit</td>
<td>146 5 161 5 0 0 0.10 0.01</td>
</tr>
<tr>
<td>bnum</td>
<td>163 5 178 5 0 0 0.7037 0.1353</td>
</tr>
<tr>
<td>city</td>
<td>71 178 178 1 0 0.6 0.4</td>
</tr>
<tr>
<td>min</td>
<td>98 10 10 10 0 0 0 0 0</td>
</tr>
</tbody>
</table>
```

Set 2 of paramnf.txt

Set 3 consists of one line for each set of comparison or matching variables. Each line must include the following information in the order given. The parameters must be separated by spaces.

- Name of matching variable. This parameter must be exactly 20 character positions long. If the name of the matching variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the first comparison field label is last in the example below)

- Starting position of the matching variable in File A. (the field starts in position 200 in File A in the first line of Set 3 in the example)

- Length, in characters, of the matching variable in File A. (13 characters long in the first line of Set 3 in the example)

- Starting position of the matching variable in File B. (the field starts in position 215 in File B in the first line of Set 3 in the example)

- Length, in characters, of the matching variable in File B. (13 characters long in the first line of Set 3 in the example)

- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (the null-data switch is set to 0 in the first line of Set 2 in the example)

- Type of comparison (wo is used in the first line of Set 3 in the example) Use one of the following eight comparisons:
  - c exact comparison: the variables being matched must agree character-by-character in order to receive the full agreement weight
  - p prorated comparison by 15 percent for numeric: for a quantitative variable such as age, this method assigns the full agreement weight if the values are within 1 but adjusts it downward in a linear manner if the difference in the two variables is 15 percent or less of the lower value; if the difference is greater than 15 percent, the full disagreement weight is assigned
  - y calls a feature developed for a specific project; should not be used
  - wo ordinary string comparator/weighting
  - uf special string comparator/weighting for first name
  - ul special string comparator/weighting for last name
  - un special string comparator/weighting for numeric address component
  - ou modified ordinary string comparator/weighting

One of the following four codes may be used with the comparison types above (those are discussed further on the following page):

- l inversion: if the current fields are not alike, compares the field with the next field on the list to see if the fields have been inverted and, if they match, assigns the same weight as if they had been in the same order
6.0 RECORD MATCHING PROGRAM

Comparison of multiple fields: The "inversion," "cross," and "critical" comparisons have been designed for some special situations.

**Inversion comparison.** The "I" or "l" options allows users to compare a field with the next field on the list to see if the fields have been inverted. Imagine that the last names in each file are compared and a corresponding weight is assigned.

```
last 1 12 1 12 0 ust 0.99233 0.32432
first 14 12 14 12 0 ust 0.99231 0.21423
```

However, there exists the possibility that the data in the two fields were inverted. For example, last name "Edward" and first name "Porter" would appear in the other files as last name "Porter" and first name "Edward."

By adding the "y" option following the "uo," the program compares the "last" field against the "first" field. This inversion comparison uses the same matching methodology as the first field match; that is, a "uo" uses the string comparator. A "cl" parameter specifies an exact match. If the inversion code is a lower-case letter (i), a match between the first field on one record and the second field on the other record is assigned a weight of one-half of the weight specified for the first matching variable. If the inversion code is a capital letter (I), a match between the first field on one record and the second field on the other record is assigned the full weight for the first variable.

**Cross comparison.** In this type of comparison, if all of the variables marked as cross variables match, then their matching weight is doubled. If only some of the cross variables match, then only the associated weights for those variables in agreement are assigned. Suppose that a user discovers that if two or more particular fields match it increases the probability of a match; for example: "first name," "month of birth," "day of birth," and "year of birth." The following is an example of the parameters in Set 3 for these four matching variables:

```
first 14 12 14 12 0 ust 0.99231 0.21421
month 28 28 2 0 ox 0.93932 0.2132
day 30 2 30 2 0 ox 0.93232 0.343
year 32 4 32 4 0 ox 0.9 0.2
```

If the four matching variables were specified as "uo" and "c": instead of "uo" and "ax," and all four matched, the matching weight would only be increased by the variable weights associated with those variables. However, these variables are designated by the "x" extension as "cross comparison" variables. Therefore, if all four match, the total matching weight will be augmented by twice as much; once for matching like any other matching variables, and once more because these variables are designated as "cross comparison" variables.

**Critical comparison.** If one of the variables in a cross match is designated as "critical" by replacing the "x" code with "k," then only three of four or more cross match fields have to

...
RECORD LINKAGE SOFTWARE

6.0 RECORD MATCHING PROGRAM

agree for the weights of the agreeing fields to be doubled as long as one of the agreeing fields is the critical field. For example, the following parameters might be included in Set 3:

- first 141214120 win 0.99231 0.21421
- month 2828200 cx 0.93032 0.2132
- day 3030200 cx 0.93232 0.343
- year 3243240 0 cx 0.9 0.2

If first name, month of birth, and year of birth agreed, the matching weight would be increased by the sum of the weights for those three variables twice, once for ordinary matching and once again because the variables are cross-match variables and include the critical variables.

Set 4 is present only if the third variable of Set 1 contains a 1. The sequence field allows an individual to check matching results which are brought together as pairs against the records in the original file being matched. It consists of the following specifications, separated by spaces.

- Name of sequence variable. This parameter must be exactly 20 character positions long. If the name of the sequence variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters.
- Starting position of sequence variable on File A.
- Length, in characters, of sequence variable on File A.
- Starting position of sequence variable on File B.
- Length, in characters, of sequence variable on File B.

Set 4 is rarely used, so it is not shown in the example.

Set 5 contains the high and low cutoffs for the match, if this option is being used. If the fourth parameter in Set 1 is equal to 1 then this option is being used. It consists of two specifications.

- High cutoff value, the score below which you are certain very few correct linkages exist. (2.2 in the example below)
- Low cutoff value, the score above which you are certain few linkages exist. (-0.1 in the example)

Pairs of records with scores between these two values must be regarded individually to determine whether they are matches or residua.

If the probabilities are assigned by logistic regression, the probability of a match linkage is the product of the probabilities of each score. For example, the probability of a score of 15.5 is correct equals 0.939. In other words, there is a 93.9 percent confident that a match with a score of 15.5 is correct.
Set 9 consists of one line that contains the high cutoff and low cutoff values for printing. If the eighth parameter in Set 1 is equal to 1, then this option is being used. It consists of two specifications:

- High cutoff value, the score above which you do not wish to view any records from the match. (35.0 in the example below)
- Low cutoff value, the score below which you do not wish to view any records from the match. (–14.0 in the example)

These values may be equal to the match cutoff values, but do not have to be. The pairs with a positive match weight will be output to prnd1.dat and the pairs with a negative weight will be output to prnd2.dat.

Set 9 of parmnf.txt

If the program fails to run properly, error messages will be in summf.dat. Error messages typically relate to increasing the size of the data structures that hold data.

6.3 Running the Program

Once the required parameter files have been created, you are ready to run the program. Before running MATCHER.EXE, make sure the following input files are in the directory:

- *parmnf.txt*: This file contains the name and location of the files to be matched.
- *prnd1.dat*: This file contains all of the parameters necessary to perform the match.
- *sorta.dat*: The sorted and standardized records from File A.
- *sortb.dat*: The sorted and standardized records from File B.
- *MATCHER.EXE*: The matching program has to be present in this directory.

Run MATCHER.EXE. It will produce the following files:

- *summf.dat*: This is the summary statistics file. It contains information on:
  1. The number of records subjected to match from both files.
  2. The total number of pairs examined.
  3. The number of records from each file that could not be paired to a record in the other file.
  4. The number of linked pairs for which the matching score exceeds the high cutoff parameter of Set 5 in parmnf.txt (i.e., matched pairs).
  5. The number of linked pairs for which the matching score falls between the high cutoff and the low cutoff parameter of Set 5 in parmnf.txt (i.e., clerical review pairs).
  6. The sizes of the largest three blocks in each match.

- *prnd1.dat* and *prnd2.dat*: These files identify the best linkage from File B to File A. Each record in the file contains the following information in the following order:
  1. The score;
  2. The block identifiers;
  3. The value of each matching variable from File A (listed in the order that the matching variables are listed in Set 3 of the parameter file, parmnf.txt);
  4. The value of each matching variable from File B (listed in the order that the matching variables are listed in Set 3 of the parameter file, parmnf.txt).

Prnd1.dat contains the content for a list of pairs with high scores (positive scores);
prnd2.dat contains the content for a list of pairs with low scores (negative scores).
6.0 RECORD MATCHING PROGRAM

6.4 Interpreting the Output

To interpret the output, you need to sort the output files from the matching run, and then print the files in a readable format.

The files need to be sorted in descending order (based on the weights from the matching run). QSORT, which had been used previously, is unsuitable for this purpose (i.e., QSORT does not have the capability to sort in descending order). For this sort, we are recommending the DOS sort utility. An example of the command needed to execute the sort utility is as follows:

```
SORT /R < PRNTD1.DAT > PRNTD1.SRT
```

This command is made up of six components:

- **SORT**
  - This command calls the DOS sort utility.
- **/R**
  - This component instructs the sort utility to sort in descending order.
- **<**
  - This instructs DOS that the name of the input file is to follow.
- **PRNTD1.DAT**
  - This is the name of the input file (which is the output file from the MATCHER program that contains matches with a positive weight).
- **>**
  - This instructs DOS that the name of the output file is to follow.
- **PRNTD1.SRT**
  - This is the name of the output file that will be created from the sort utility.

After the files have been sorted in descending order, you are ready to run PRINTER.EXE.

Before running PRINTER.EXE, make sure the following input file is in the directory:

- **prnt1.srt**

An example of the command needed to execute PRINTER.EXE is as follows:

```
PRINTER < PRNTD1.SRT > PRNTD1.RPT
```
Before running RESID2.EXE, make sure the following input files are in the directory:

- sorta.dat  Consists of the cases in the original File A after standardization and sorting. This file was created when the standardized file was sorted by the blocking variables; see Section 5.0.
- sortb.dat  Consists of the cases in the original File B after standardization and sorting. This file was created when the standardized file was sorted by the blocking variables; see Section 5.0.
- parma.txt  Specifies the names and locations of sorta.dat and sortb.dat.
- parmfst.txt Described in Section 6.2.
- ptnmf.dat  Created by MATCHER.EXE.
- cutoff.dat  As explained above.

Run RESID2.EXE.

The following files contain the non-matches, which will be sorted on new blocking variables and re-matched on a subsequent run of the matching program.

- r_sorta.dat
- r_sortb.dat

After sorting, they are ready for a second matching attempt. See Section 5.0 for more information about matching. The second match should be run in a different directory from the first. Otherwise, the old versions of several files will be lost.

### 6.6 Executing the Second and Subsequent Matching Runs

For a second or subsequent matching run, four input files are required:

- r_sorta.dat  This is the sorted residual File A (i.e., those records from File A that did not match on the first match).
- r_sortb.dat  This is the sorted residual File B (i.e., those records from File B that did not match on the first match).
- parma.txt  Specifies the names and locations of the input files.
- parmfst.txt  This file contains the necessary parameters to run the match.

Rerun MATCHER.EXE by repeating the steps beginning in Section 6.1.
7.0 ADVANCED AUTOMATIC ADJUSTMENT FEATURES

The advanced automatic adjustment features allow additional control over the matching process by users who are well versed in the mathematical theory of record matching.

7.1 Creating Sorted Data Files

Rename the first sorted data file as \textit{sorta.dat}.

```
| ab11ehrt | ah | 25000001 |
| ab21ehrt | ah | 25000002 |
| ab31ehrt | ah | 25000003 |
| ab41ehrt | ah | 25000004 |
| ab51ehrt | ah | 25000005 |
| ab61ehrt | ah | 25000006 |
| ab71ehrt | ah | 25000007 |
| ab81ehrt | ah | 25000008 |
| ab91ehrt | ah | 25000009 |
| ab101ehrt | ah | 25000010 |
```

Example of \textit{sorta.dat}

Rename the second sorted data file \textit{sortb.dat}.

```
| ab11eohrt | ah | 25000001 |
| ab21eohrt | ah | 25000002 |
| ab31eohrt | ah | 25000003 |
| ab41eohrt | ah | 25000004 |
| ab51eohrt | ah | 25000005 |
| ab61eohrt | ah | 25000006 |
| ab71eohrt | ah | 25000007 |
| ab81eohrt | ah | 25000008 |
| ab91eohrt | ah | 25000009 |
| ab101eohrt | ah | 25000010 |
```

Example of \textit{sortb.dat}

The input files must be sorted by the logical blocking criteria given in the second set of parameter file lines (see Section 7.2 below, under "Set 2"). If they are not, the program will stop with an error message indicating which record in which file is out of sort order.
7.2 Creating the Parameter File parm.c.txt

Create the file parm.c.txt, which contains the set of input parameters. This file allows the specification of additional parameters, beyond those included in parmnn.f.txt, which was described above in 6.3. Change your parameters as required.

An example of parm.c.txt is shown below. The labels "Set 1," "Set 2," and "Set 3" are not part of the file but are used for reference. As for parmnn.f.txt, the maximum record size is 499 and the maximum size of a single field is 60 characters.

```
Set 1
  1 3 120 120
Set 2
  5 20 5 20 0 u 0.95 0.25
  25 10 25 10 0 u 0.80 0.20
  40 3 40 0 0 0.90 0.30

Example of parm.c.txt

Set 1 consists of a single line containing four parameters:

- Number of blocking fields. (there is 1 blocking field in the example below, so the program will interpret only one line as containing information about the blocking variable)

- Number of matching variables. (there are 3 comparison fields in the example, so the program will interpret three lines as containing information about the matching variables)

- Length, in characters, of records in File A. (120 characters long in File A in the example)

- Length, in characters, of records in File B. (120 characters long in File B in the example)

Set 1 of parm.c.txt
```

Set 2 of parm.c.txt

Note that the blocking field may be part of the field occupied by a variable. For example, if the last name field began in position 1, the above parameters would specify blocking on the first four characters of the last name. The last names SMITH, SMITHers, SMITHson, and SMITHs would all be in the same block; the last names ALBEnis, ALBErghetti, ALBErt, and ALBErtson would also be in the same block.
Set 3 consists of one line for each matching variable. The number of lines must correspond to the number in the second parameter in Set 1. Each line must include the following parameters:

- **Name of the matching variable.** This parameter must be exactly 30 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (The first comparison field label is last in the example below)
- **Starting position of the matching variable in File A.** (The field starts in position 5 in File A in the example)
- **Length, in characters, of the matching variable in File A.** (20 characters long in File A in the example)
- **Starting position of the matching variable in File B.** (The field starts in position 5 in File B in the example)
- **Length, in characters, of the matching variable in File B.** (20 characters long in File B in the example)
- **Null-data switch.** This parameter is currently nonoperational; use 0 as a place-holder. (The null-data switch is set to 0 in the first line of Set 2 in the example)
- **Type of comparison (u is used in the first line of Set 3 in the example):**
  - exact
  - prorated comparison by 15 percent for numeric
  - calls a feature developed for a specific project, should not be used
  - ordinary string comparator/weighting
  - special string comparator/weighting for first name
  - special string comparator/weighting for last name
  - special string comparator/weighting for numeric address component, modified ordinary string comparator/weighting
- **An estimated probability #1.** (0.95 in the first line of Set 3 in the example)
- **An estimated probability #2.** (0.25 in the first line of Set 3 in the example)

```
  3 120 120
  lsk 1 4 1 4 0
  last 5 20 5 20 0 0 0.95 0.25
  first 25 10 25 10 0 0 0.80 0.20
  last 40 20 31 0 0.70 0.30

Set 3 of parmu.txt
```

If the program fails to run properly, error messages will be in chke.dat.
RECORD LINKAGE SOFTWARE 7.0 ADVANCED AUTOMATIC DIJESTMENT FEATURES

Set 1 consists of a single line containing four parameters.

- Number of blocking fields. (there is 1 blocking field in the example below, so the program will interpret only one line as containing information about the blocking variable)

- Number of matching variables. (there are 3 comparison fields in the example, so the program will interpret three lines as containing information about the matching variables)

- Length, in characters, of records in File A. (120 characters long in the example)

- Length, in characters, of records in File B (120 characters long in the example)

Set 1

<table>
<thead>
<tr>
<th></th>
<th>1 2 120 120</th>
</tr>
</thead>
<tbody>
<tr>
<td>blk</td>
<td>1 1 4 1</td>
</tr>
<tr>
<td>last</td>
<td>5 20 5 20 0</td>
</tr>
<tr>
<td>first</td>
<td>25 10 25 10 0</td>
</tr>
<tr>
<td>now</td>
<td>40 3 60 71</td>
</tr>
</tbody>
</table>

Set 1 of parmu.txt

RECORD LINKAGE SOFTWARE 7.0 ADVANCED AUTOMATIC DIJESTMENT FEATURES

Set 2 consists of a single line for each blocking variable. The number of lines must correspond to the number in the first parameter in Set 1. Each line contains the following parameters:

- Name of the blocking variable. This parameter must be exactly 20 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (the blocking field label is blk in the example below)

- Starting position of the blocking variable in File A. ("blk" starts in position 1 of File A in the example)

- Length, in characters, of the blocking variable in File A. (4 characters long in the example)

- Starting position of the blocking variable in File B. ("blk" starts in position 1 of File B in the example)

- Length, in characters, of the blocking variable in File B. (4 in the example)

Set 2

<table>
<thead>
<tr>
<th></th>
<th>1 2 120 120</th>
</tr>
</thead>
<tbody>
<tr>
<td>blk</td>
<td>1 1 4 1</td>
</tr>
<tr>
<td>last</td>
<td>5 20 5 20 0</td>
</tr>
<tr>
<td>first</td>
<td>25 10 25 10 0</td>
</tr>
<tr>
<td>now</td>
<td>40 3 60 71</td>
</tr>
</tbody>
</table>

Set 2 of parmu.txt
Set 3 consists of one line for each matching variable. The number of lines must correspond to the number in the second parameter in Set 1. Each line must include the following parameters:

- Name of the matching variable. This parameter must be exactly 20 characters long. If the name of the blocking variable is less than 20 characters, extra spaces must be included at the end to make a total of 20 characters. (The first comparison field label is last in the example below)
- Starting position of the matching variable in File A. (The field starts in position 5 of File A in the example)
- Length, in characters, of the matching variable in File A. (20 characters long in File A in the example)
- Starting position of the matching variable in File B. (The field starts at position 5 in File B in the example)
- Length, in characters, of the matching variable in File B. (20 characters long in File B in the example)
- Null-data switch. This parameter is currently nonoperational; use 0 as a place-holder. (The null-data switch is set to 0 in the first line of Set 2 in the example)

<table>
<thead>
<tr>
<th>Set 3</th>
<th>1 2 120 120 1 4 1 4 5 20 5 20 0 25 10 25 10 0 40 3 40 3 1</th>
</tr>
</thead>
</table>

Set 3 of parmu.txt

If the program fails to run properly, error messages will be in checku.dat.

7.5 Running the U-Probability Parameter Generation Program

Run U1. This program provides an alternative method for obtaining u-probabilities that can be used in checking the u-probabilities produced by the EM program. The EM program produces both m- and u-probabilities. The u-weight program U1.C produces random-agreement probabilities that are often close to the u-probabilities from the EM program.

The u-probability is the probability that the content of a pair of fields agree, given that the records being compared do not match. Effectively, it is the probability that the content of the fields agree at random. This program will produce file uprob.dat.

7.6 Running the Expectation-Maximization (EM) Program

Create parmem.txt. Parameters include the number of fields (variables), an upper bound on the proportion of pairs estimated as matches, the epsilon used in fitting, and the maximum number of allowable iterations.

Example of parmem.txt

Ran EM. This refers to two files generated by C2, enc.dat and parmem.txt. This program generates more accurate probabilities. EM will produce two files:

- enc.dat
- enc.ark
APPENDIX: RUNNING A MATCH ON A SAMPLE FILE WITH TWO PASSES

This appendix describes a sample matching job run against two sample files that SRD had compiled from publicly-available data about celebrities. Because the data were from public sources, there are no confidentiality issues in their use. The data consist of names and addresses only. One file contained 1,161 records; the other file contained 1,600 records. Names and addresses had already been standardized.

A.1 General Description of Sample Matching Job

The celebrity data were put through a two-pass match. In the first run of the matcher against the two files, 279 matched pairs and 17 clerical review pairs were found. Clerical review pairs are pairs with matching scores between the high and low cutoff values. When the residuals (919 records from the file of 1,161 and 1,357 records from the file of 1,600) were passed through a second match using new blocking and new cutoff values, it picked up 64 new matches and 44 clerical review pairs. This illustrates the need to block in different ways in order to pick up all the matches.

Three directories were used, two for the matching runs, and one for the residuals run. This made it easier to keep track of things, and avoided having to rename files from one run in order to get out of the way of the next run and preserve results, a problem if one tries to do all of this in one folder.

The three directories in this case were named SLn (for State and Last initial), Resid, and fcfncl (for first character first name and first character last name), viz.
A.2 Setting Up the Matching Run

The data files were in another directory used for standardization and had been sorted by state and first letter of last name. These are pointed to by jsrmnt.txt, which contains:

D:\match\stanoutB.srt
D:\match\stanoutA.srt

If these files had been copied into the Sml directory instead, the "D:\match\cell" part of the path could be dropped and the entire directory contents would be portable to any other compatible machine and directory without change.

For the first match run, the parameters in jsrmnt.txt were:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>290</td>
<td>0.000</td>
</tr>
<tr>
<td>100</td>
<td>0.000</td>
</tr>
<tr>
<td>214</td>
<td>229</td>
</tr>
<tr>
<td>st</td>
<td>93.2</td>
</tr>
<tr>
<td>103</td>
<td>2.0</td>
</tr>
<tr>
<td>fc</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>215</td>
</tr>
<tr>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>0.000</td>
</tr>
<tr>
<td>13</td>
<td>0.000</td>
</tr>
<tr>
<td>215</td>
<td>0.000</td>
</tr>
<tr>
<td>13.0</td>
<td>0.000</td>
</tr>
<tr>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>0.9799</td>
<td>0.0400</td>
</tr>
<tr>
<td>0.9840</td>
<td>0.0078</td>
</tr>
<tr>
<td>0.9490</td>
<td>0.04</td>
</tr>
<tr>
<td>0.8438</td>
<td>0.2354</td>
</tr>
<tr>
<td>0.6630</td>
<td>0.2332</td>
</tr>
<tr>
<td>0.40</td>
<td>0.01</td>
</tr>
<tr>
<td>0.7037</td>
<td>0.1533</td>
</tr>
<tr>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>0.67</td>
<td>0.55</td>
</tr>
<tr>
<td>0.55</td>
<td>0.55</td>
</tr>
<tr>
<td>2.2</td>
<td>0.1</td>
</tr>
<tr>
<td>35.0</td>
<td>3.40</td>
</tr>
</tbody>
</table>

The cutoffs 2.2 and -0.1 were reasonable values; not optimized by experience with the data. If a file named init.dat had been present due to using the EM facility described in Section 7.8, the values in that file would have overridden the values from jsrmnt.txt seen here. In this sample run, no init.dat file was employed.

Note that the matching fields include "zip" with U.5 specified for both the match and non-match probabilities. This gave "zip" no role whatsoever in the matching scoring, but forced it to appear in the output files, preml.dat and preml2.dat.

A.3 Running the Matcher

Executing MATCHER.EXE produced the summary file, summfmt.dat. This was accomplished by the following two commands:

```
Match fields [D:\match\stanoutB.srt] with [D:\match\stanoutA.srt].
```

A.4 Reviewing the Output and Choosing Cutoffs

Sorted the files preml.dat and produced preml.srt. File preml.srt contained the score, the blocking value, and the specified matching field values from both celebrity files in each of its records. This rather long record is not appropriate for review by human eye. Therefore, we ran PRINTEXE to reformat the output to preml.dat in a way suitable for printing and review. PRINTEXE replaces the title character (=) with a carriage return so that each record is displayed as three lines.

```
sort pr <preml.dat >preml.srt
```
We created the file cutoff.dat as one line containing the cutoffs determined in the foregoing; that is:

```
3.0 1.0
```

Again, parma.tar listed the input file names. The file pmrfd.dat, produced by the matcher, was copied from the snl folder. Then we ran RESID2.EXE to produce the smaller file (non-matches) to be re-blocked and re-matched. The outputs from RESID2.EXE were r_sorta.dat and r_sortb.dat.

A.6 Setting Up the Second Matching Run

First we copied r_sorta.dat and r_sortb.dat from the Resids folder to the fcrfl database folder shown below. These files had to be sorted by first character of the first name with first character of last name. In File A these were in columns 174 (fcf) and 200 (fcf). In File B they were in 189 and 215. The sort produced r_sorta.srt and r_sortb.srt.

```
<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Type</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcf1.dat</td>
<td>1207</td>
<td>BCF</td>
<td>12/28/1980 12:53 PM</td>
</tr>
<tr>
<td>fcf2.dat</td>
<td>1207</td>
<td>BCF</td>
<td>12/28/1980 12:53 PM</td>
</tr>
</tbody>
</table>
```

So our choice for the lower cutoff value was 1.0.

A.5 Running the Residuals

Now we were ready to run the residuals program in the Resids folder:

```
<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Type</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resid.exe</td>
<td>10258</td>
<td>BCF</td>
<td>12/28/1980 12:53 PM</td>
</tr>
<tr>
<td>Parma.dat</td>
<td>118</td>
<td>BCF</td>
<td>12/28/1980 5:05 PM</td>
</tr>
<tr>
<td>pmrfd.dat</td>
<td>978</td>
<td>BCF</td>
<td>1/5/1981 5:27 PM</td>
</tr>
<tr>
<td>pmrfd.dat</td>
<td>118</td>
<td>BCF</td>
<td>1/5/1981 12:25 PM</td>
</tr>
<tr>
<td>pmrfd.dat</td>
<td>1593</td>
<td>BCF</td>
<td>1/5/1981 12:29 PM</td>
</tr>
<tr>
<td>pmrfd.dat</td>
<td>3059</td>
<td>BCF</td>
<td>1/5/1981 12:48 PM</td>
</tr>
</tbody>
</table>
```

```
(0 objects) 8389 (Disk free space: 1.035G)
```
RECORD LINKAGE SOFTWARE

APPENDIX: RUNNING A MATCH ON A SAMPLE FILE WITH TWO PASSES

For the second match run, the parameters in paramf.dat were:

2 10 0 100 0 1 0 2 14 2 20
fct 174 1 189 1 0
fin 200 1 215 1 0
last 200 13 215 13 0 un 0.9799 0.0430
first 174 12 189 12 0 un 0.9840 0.0078
mid 187 12 202 12 0 un 0.49 0.04
hsm 187 5 117 5 0 c 0.8439 0.2354
srt 146 1651 5 0 c 0.10 0.01
unit 163 5 176 5 0 c 0.7037 0.1353
city 71 17 81 17 0 c 0.6 0.4
state 93 2 103 2 0 c 0.5 0.5
zip 96 5 106 5 0 c 0.5 0.5
2.2 -0.1
35.0 -14.0

A.7 Running the Matcher—Second Pass

Executing MATCHER.EXE produced the summary file, summf.dat, viz.:

Matching files [r_sorta.srt] with [r_sortb.srt]

No init file
Blocking Field: fct
Matching Field: last agree: 3.1263 disagree: -3.8631
Matching Field: first agree: 4.8375 disagree: -4.1273
Matching Field: mid agree: 2.9252 disagree: -6.6325
Matching Field: hsm agree: 1.2766 disagree: -1.5882
Matching Field: srt agree: 0.6868 disagree: -0.8684
Matching Field: unit agree: 2.3026 disagree: -0.0953
Matching Field: bann agree: 1.6489 disagree: -1.0710
Matching Field: city agree: 0.4055 disagree: -0.4055
Matching Field: state agree: 0.0000 disagree: 0.0000
Matching Field: zip agree: 0.0000 disagree: 0.0000

MATCHING SUMMARY STATISTICS

919 records in the First file.
1357 records in the Second file.
5439 pairs of records compared.
93 records skipped in the First file.
188 records skipped in the Second file.
64 matched pairs.
44 clerical review pairs.
717 unmatched records in the First file.
1060 unmatched records in the Second file.
0 blank records in the First file.
0 blank records in the Second file.

This concluded the sample run.