John Blodgett
Urban Information Center
University of Missouri
St. Louis, Mo. 63121

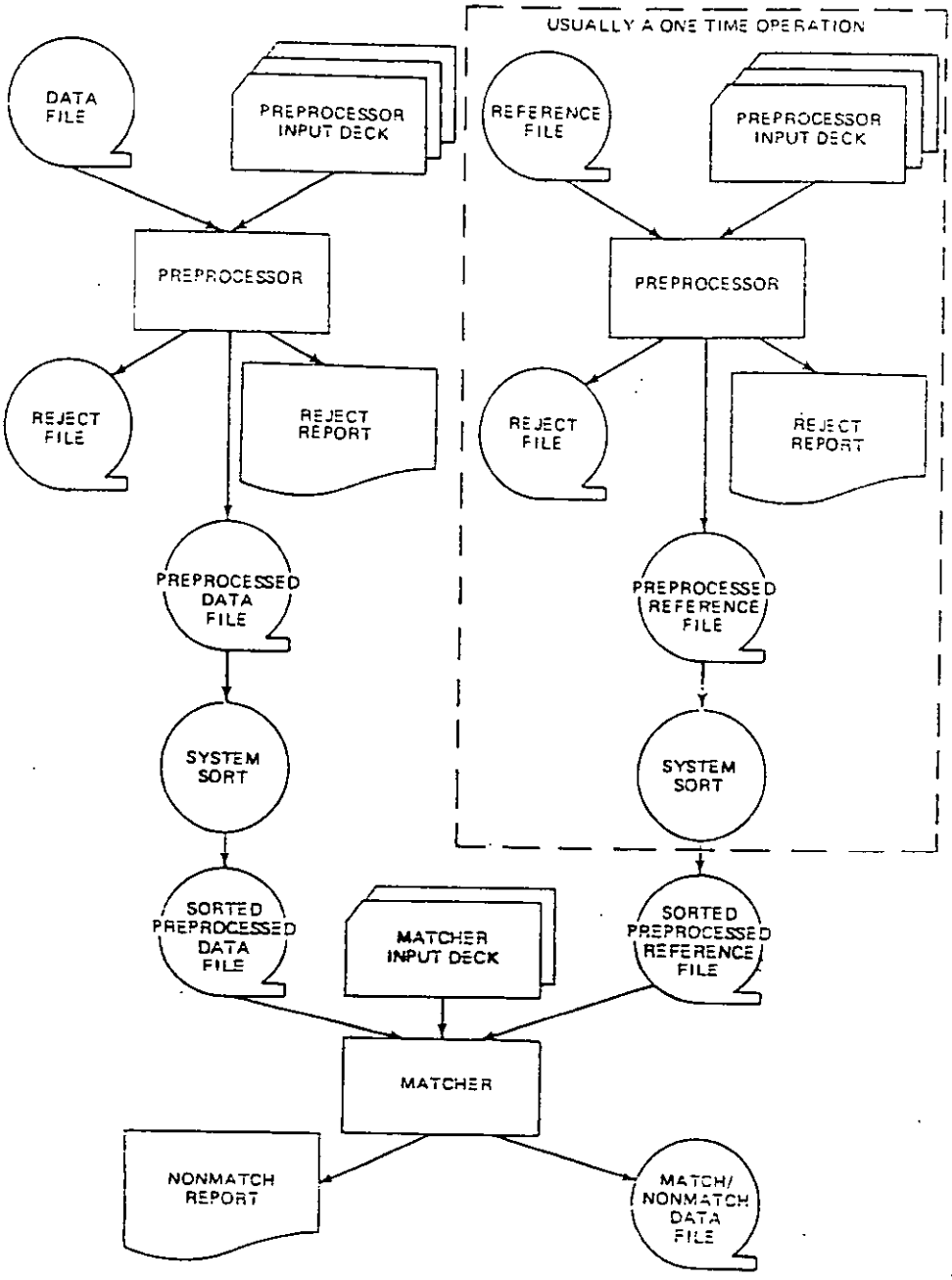
A PROGRAMMER FRIENDLY APPROACH TO GEOCODING

ABSTRACT. Geocoding - the process of linking data files containing street addresses to a master street reference file - is a problem which has been "solved" with the Census Bureau's DIME/ADMATCH (or UNIMATCH) system. However, the value of these extremely useful tools is diminished somewhat because of the complexity of the programming involved in applying them. There are perils involved in creating your reference ("nickel") file using DIME and ADMATCH/PREP. Mistakes are easy to make and hard to find. More serious are the problems involved in applying the system to a wide variety of files in a timely, efficient and reliable manner. The system developed by the UMSL Urban Information Center provides a programmer familiar with the general workings of ADMATCH a set of powerful software tools that can greatly simplify the complex and error - prone data management aspects of using ADMATCH. These tools are provided in the form of SAS ® preprocessor macros, and are written with total flexibility and ease of use (by a programmer familiar with the SAS language) in mind. Because they are in source code format and because they rely heavily on easily modified "default" parameter values, they can very easily be customized to fit the needs of a wide variety of users.

INTRODUCTION

Figure 1 is a reproduction of the ADMATCH system flowchart as it appears in the ADMATCH User's Manual. While it may be a useful depiction of the basic steps involved in geocoding with ADMATCH, it is a very partial depiction of most real life applications of the software. It does not show, for example, any program or subsystem that does anything about preprocessor rejects or nonmatched address records. They are simply written to a file and/or listed in an exception report. What happens to them next is beyond the scope of the ADMATCH software. But in terms of a practical geocoding system, an efficient and relatively easy-to-use subsystem for fixing rejected and nonmatched records is critical for many if not most applications.

The flowchart also omits the "post-processing" phase (this is covered in a separate manual for some reason) in which the various matched, nonmatched and rejected records are all brought back into



ADMATCH SYSTEM

FIGURE 1

one or more final output files. In the portion of the flowchart dealing with preprocessing the "reference file" (presumably a DIME file or its equivalent) there is no indication of the need for a special preprocessor step to split the 2-sided "DIME" records into single-sided "nickel" records prior to preprocessing. There is also no indication that more than one system sort may be required in order to produce multiple versions of the reference file for matching or different criteria (e.g. one version for matching on ZIP, another for matching on state and place.)

Our purpose here is not to be critical of the ADMATCH system or its authors. We think the basic ADMATCH program modules, the preprocessor and the matcher are two of the most clever, useful and reliable software products ever written. However we feel that as a software system for dealing with the entire complex problem of geocoding - especially in an environment requiring a lot of flexibility as well as computer resource economy and where programmer time is at a premium - ADMATCH simply does not do enough. What we would like to do in this paper is to discuss an approach to geocoding that retains the valuable program modules from ADMATCH, but incorporates them into a more complete geocoding system. The system we have developed has been very much customized to fit the needs of our own environment, and may not be totally appropriate for another shop (many shops could undoubtedly do with something a lot simpler.) What we would really like to emphasize here is not the specifics of our system, but rather the underlying approach or philosophy behind it.

DESIGNING THE GEOCODING SYSTEM: ASSUMPTIONS AND REQUIREMENTS

In designing a "complete" geocoding system (or any other system for that matter) it is a good idea to start with a list of features and capabilities that you would like your system to have. The following list describes the major functions we wanted our system to be able to perform, and includes some specific guidelines on how we wanted these capabilities implemented. The list reflects our assumptions and rather subjective evaluations ("filling out the preprocessor control card is tedious and error-prone"). It is based on several years experience running ADMATCH software without a structured set of software modules to complete the system.

1. The process of filling out the PREP (ADMATCH preprocessor) control card is tedious and error-prone. The errors are usually not detected (you pointed to the wrong column but PREP didn't know the difference), and often result in complete runs which are a waste of time and computer resources. Control cards used in other modules, especially SAVE cards in the

matcher module, are fraught with similar perils. There is far too much column-counting going on.

2. A system for displaying unmatched addresses and providing for updating of these rejects is required.

3. An ADMATCH project creates too many datasets and requires far too much coding of error-prone JCL statements. Our system should standardize as many files as possible so that the DD statements can be stored as part of a JCL procedure, and never have to be coded by the user. A more systematic way of dealing with all the permanent datasets is required.

4. As a record passes through the geocoding process it can appear on many different files: the original input, the preprocessed file, the sorted preprocessed file, the initial match run reject file, the sorted match run reject file, the updated match run reject file, the preprocessed (again) updated match run reject file, etc., etc. It is not uncommon for these records to be several hundred characters long, and for the files to contain thousands, tens of thousands and occasionally even hundreds of thousands of records. Since ADMATCH looks at only a small part of each record in order to standardize and match it, this information should be stripped off and only these relatively small records should ever be processed by any but the first and last geocoding system modules. This should result in significant savings in storage and processing costs.

5. It can be very expensive to pass a relatively small address file (of say, a few thousand records) against our full 200,000 + - record reference file (the St. Louis SMSA preprocessed "Nickel" file.) It would be a significant improvement if we could do a "pre-check" of the addresses against a much smaller disk-resident reference summary file, to see which records can be matched at the ZIP and/or preprocessed street name level, and to be able to correct these "won't match" records before actually doing a match run against the full tape file. At the same time, this would be a good place to use an "alias file" to substitute for common misspelling or true alias street names.

6. The subsystem for going from the DIME file to a preprocessed nickel file should be streamlined and made more flexible. It would be nice not to have to preprocess every DIME record twice, i.e. to preprocess the actual DIME record prior to splitting into 2 block face records. The subsystem should allow for creating the compact summary datasets at the ZIP and PREPNAME levels, and for printing out compact well labeled reports that can be used to assist in looking up unmatched records.

7. Programmers in our shop do almost all their work using SAS. It would therefore be extremely beneficial if our geocoding system could at least begin and end with SAS datasets. This would not only save us from having to write PUT/INPUT statements but would go a long way towards solving the data management problems referred to under item 3, above.

8. Programmers and self-reliant users should be able to set up and run simple geocoding applications without having to read the pink and purple ADMATCH manuals.

A SAS-BASED SOLUTION: WHY AND HOW

We chose the SAS software package as the basis for our geocoding system, not because of any of its considerable statistical capabilities, but because of its excellent facilities for sequential file management and powerful programming language. When we first began coding the system (around 1979) we were not particularly impressed with the capabilities for writing "user-friendly" systems in SAS, but felt that its macro features would be adequate. Our first SAS-based system, written using "old style" SAS macros was a major improvement over what we had been doing, but was not as flexible and easy to use as we would have liked. It worked off a lot of "parameter macros", and you wound up going in and modifying the actual source code a lot to handle "special" cases. The system was then rewritten (or more accurately, is still in the process of being rewritten) with the introduction of the much more powerful SAS preprocessor macro language which became available in late 1982.

URISA imposes time and space requirements that mercifully prohibit any detailed description of how the SAS/ADMATCH system works. But here are some of its more notable features:

1. The system is driven by a set of "programmer friendly" macros that allow the user to supply parameter values which are used by the SAS preprocessor to (sometimes conditionally) generate the appropriate SAS statements.

2. The ADMATCH system modules PREP and MTCH (the actual load modules, unaltered for over 10 years) are invoked from inside SAS. In terms of CPU time these two modules still do most of the work.

3. All files passed to the PREP program are in exactly the same format. Each record consists of a 7-digit numeric KEY field, a 33-character ADDRESS field and a 5-character ZIP field. It always has the same LRECL and blocking factor. This means that the error-prone PREP control card becomes a "constant", and does not have to be prepared anew for each run. It also means that the file which is read by PREP (SYS004) can be written by SAS to a temporary dataset that is a "constant" (JCL-wise) for all runs. Similarly, the PREP output files (SYS005 and SYS006)are "constant" temporaries that can be easily converted to SAS datasets. The user normally only needs to code one DD statement defining his SAS data library, while the 3 DD statements for SYS004, SYS005 and SYS006 become "invisible" to the user because they are coded as part of a standard JCL procedure. Similar savings occur in all subsequent phases. Storing files as SAS datasets makes them more compact and much easier to sort, list, document, delete, edit, merge, summarize and generally take care of.

4. Preprocessing the reference (DIME) files is handled with a 3-macro subsystem. The first converts the sequential DIME file to a SAS dataset; the second preprocesses the file and stores the result as a SAS NICKEL dataset; and the final macro sorts and writes a specific sequential nickel file. (The second module has options to generate reference SAS datasets of all ZIPs and ZIP/PREPNAME combinations.)

5. It is still possible to get 37% match rates in high growth areas, but you get such nice concise informative reports depicting your failures.

6. This system is rarely used to do the same thing twice. Yet, we can usually give 48-hour turnaround (assuming no reject lookups) with almost total reliability that the geocodes will be where you wanted them on your output file or dataset (converting final results back to sequential files instead of SAS datasets is not a problem.)

7. Programmers can be taught how to use the system in a day or two. They do not need to know all the details of the PREP and MTCH program logic, although it is important that at least one person in the shop be knowledgeable in these areas.

8. Many geocoding applications have as their final goal the linking of aggregated geocoded files with census data for market penetration reports. For example, after you attach tracts to the student data base you want to get a report showing both total students per tract and perhaps total students per tract per capita. It is also common to then

want to display this information in a computer drawn map. All of this type of postprocessing is made relatively simple with SAS and SAS/GRAPH.

9. We have not done any careful benchmarking or conducted any experiments to see how geocoding with our SAS/ADMATCH system compares to how we were doing with PL1 and IBM utilities. But our guess is that this system requires about a fifth of the programmer effort, about half of the computer resources (CPU time, tape and disk I/O, permanent data storage), and is several times more likely to do the job right the first time. Overall turnaround time has improved dramatically.

CONCLUSIONS

Geocoding is a tough problem. Even with excellent public domain software products like ADMATCH available, most shops are justifiably reluctant to get into this specialized kind of processing. You have to be willing to invest a certain amount of time in learning about the specifics of how a geocoding system works. You also are going to run into problems with very poor match rates in certain areas unless you are willing to take on the task of keeping your reference file up to date. And, finally, unless you have some very fixed requirements for what kinds of files you will be geocoding and just how you want to handle them, you should expect to spend a fair amount of time in developing auxiliary software to handle the very considerable housekeeping tasks involved in managing all the datasets you'll be creating. If you are in an environment where computer resource costs are a significant consideration, geocoding large files can get very expensive. This has resulted in a lot fewer shops actually doing in-house geocoding. They are either sending it out to a service bureau, doing it by hand, or getting by with ZIP data. This is an unfortunate waste of some good software and some very expensive DIME files.

While we certainly don't think our specific software is any universal cure for the under-geocoding phenomenon, we do think that the general approach we have taken has certainly solved a good part of the problem in our shop. This approach can be characterized as follows:

1. Avoid starting over. Somebody has probably already solved all or part of the problem. Don't waste good solutions just because they're not perfect.

2. Find a good general-purpose data management tool and build your solutions around it. SAS is an excellent tool for such

things; IBM JCL and IBM utilities are not.

3. Avoid building solutions that cannot be easily modified to handle all the special cases you'll never be able to anticipate. Avoid constants where parameters will fit nicely. But make the parameters easy to specify, with lots of defaults.

4. Instead of end-user friendly solutions, build programmer friendly solutions, that programmers can easily customize into user-friendly ones.

5. Use preprocessors or meta-languages to build your system. Then you can enjoy lots of flexibility with lots of parameterization, with almost no sacrifice in efficiency.

6. Spend most of your time worrying about exactly what you want your system to do, then find someone who can tell the computer exactly how to do it.

7. If you decide to write a paper describing your system, don't wait until two days before the manuscript is due to give it to your secretary. If you do, they will not be programmer friendly.

ACKNOWLEDGEMENTS

SAS is a registered trademark of SAS Institute, Cary, North Carolina.