OPTIMAL ORGANIZATION OF CENSUS MIGRATION FILES: A CASE STUDY

Ramom Torrecilha
William Gates

CDE Working Paper No. 87-44

OPTIMAL ORGANIZATION OF CENSUS MIGRATION FILES: A CASE STUDY

Ramom Torrecilha and William Gates.

Center for Demography & Ecology

University of Wisconsin-Madison

1180 Observatory Drive

Madison WI 53706

CDE Working Paper 87-44

Acknowledgements:

This analysis was conducted in part using facilities provided by a Population Research Center Grant (HD05876) from the Center for Population Research of the National Institute of Child Health and Human Development to the Center for Demography and Ecology.

OPTIMAL ORGANIZATION OF CENSUS MIGRATION FILES: A CASE STUDY

INTRODUCTION

Managing data sets is seldom a task that most researchers enjoy doing. More often than desired the difficulties encountered during the process are compounded by the size of the data set, poor documentation, lack of appropriate resources, and by our unfamiliarity with software. Despite all the problems that data management may cause, it still constitutes a very important stage in most research projects. We do not need to belabor the fact that the quality of data, and the way in which we choose to organize it, plays a part in our ability to answer research questions. This paper is about data methodology. It is hoped that this exercise will provide some insights about alternative ways that one can think of, and better implement the management of large data sets.

Specifically, this paper will describe the ways in which two migration files, each containing approximately 2.4 million records, were prepared for use in a research project attempting to clarify the impact of migration to chronically poor counties in the United States. However, the research questions for which these two data sets have been prepared will not be presented. In addition, we will try to be as explicit as possible about all the steps used for the preparation of the data. Familiarity with Ingres would certainly be helpful to the reader.

In the first section of this paper we will describe the data sets, and state our task. Section two will elaborate on how solutions to the problem where developed and executed. Next, we will present an alternative data structure that could be utilized to solve the data processing problem. The last section will finalize this paper with a brief conclusion.

DEFINING THE TASK

Migration, broadly and briefly defined, is the study of the movement of people between and within geographic localities. Migration is understood to be a social phenomenon that is contingent upon a number of factors such as age, ties to places of destination and origin, expectations, economic opportunities, quality of life, and many others. The study which lead to the management of two large data sets is an attempt to answer questions pertaining to the above factors, and the flow of migrants to and from economically depressed non-metropolitan counties. Contained in these two data sets is information about all possible streams of migrants to all possible destinations, and from all possible origins, in the United States. In addition, these two sets also have some basic demographic characteristics of the stream.

Each data set contains approximately 2.4 million records, with each record being 24 characters long. In each record we have data on state and county of previous and present residence, age, sex, race, and the size of the stream for 1965-70, and 1975-80. For the execution of our task, state of present residence, originally a two character variable, and county of present residence, a three character variable, were combined into one single variable of five characters, named 'dfips'. State of previous residence, and county of previous residence - which reflect migrants' residence five years ago - were also combined into a single five characters variable called 'ofips'. Age was kept as originally recorded into 17 intervals, of equal width, beginning at age of 5 up to age 85 plus. Sex and race were dichotomized into their respective categories, with black receiving the code of '1' and nonblack the code of '0'. Nine characters were reserved to represent the size of the migrant stream, and such a variable was given the name of 'weight'. However, not all the space reserved for this variable was used. For the most part the size of streams was

represented only by two digits, with the remaining space being filled with zeros. Figure 1 represents the structure and format of the data. This particular record portrays a stream of nine subjects who migrated within the State of Alabama, from Auoga County to Baldwin County. All the subjects within this particular stream fall in the 09 age interval, which encompasses those between the ages of 45 and 49 years old. They are males and nonblack.

```
|01003|01001|09|1|2|0000000009|

dfips  ofips age sex race weight
```

Figure 1: Data Structure and Variable Specification

Looking at Figure 1, one could get the impression that the data sets are generally straightforward, and the relatively short length of each record perhaps contributes to such an impression. However, a closer inspection shows that these twenty four characters have more information stored in them than meets the eye. To get a feel for the magnitude of these two sets consider the fact that the data have migration figures for all states in the country, don't forget to multiply that number by two since a migrant that leaves a state will most likely reside in another one. The same thing must be done for all counties in the United States, and remember that there are 3100 plus counties. One should also find a way of adding the various combinations for age, sex, and race. The final product should be a number well in the millions.

In spite of all these numbers, and all the information contained in them, the sad truth is that they tell us very little about migrants because only selected demographic characteristics were collected. It turned out that in order to best answer the questions proposed by the research project for which

these data sets were being prepared additional information on migrants was necessary.

Additional information on migrants, and on their places of destination and origin, was previously stored in another data set. However, the latter data set was not compatible with the two original sets. Incompatibility between these sets of data resulted from the fact that unlike the two migration files where the total number of counties were approximately 3200, the latter data set had only 3088 counties. This discrepancy in the number of counties was due to a prior classification scheme used by the research team, and not because of missing information on counties. In any event, incompatibility between the two sources of information, dictated that our task be one of aggregation so that the two migration files would in the end have 3088 counties, allowing for further merging of the files.

Aggregating records on a file is a relatively easy task to be accomplished; a number of software packages have been designed for such a purpose. However, often times existing software is not always the most efficient solution to the problem, and internal limitations in most of these packages prevent the manipulation of a large amount of records. Given these limitations, it was decided that Ingres was the appropriate tool to sort and aggregate those records, since Ingres was specially designed for processing large amounts of data using relatively small amounts of computer time. It is to the implementation of Ingres that we shall turn now.

GETTING STARTED

Before describing the actual procedures to be used for solving our task, a word about the conceptualization of what the final product is going to look like is in order.

We like to think that one's first exposure to a new data set corresponds to the experience of inspecting a new book for the first time. Driven by our curiosity, such a experience often results in a close inspection of the book's cover, a look at the table of contents, at the organization of the chapters, and perhaps a glance at the bibliography. Only later, will we reach for the book again with the intent of reading it. It has been said that the time spent learning about the organization of a new book is not wasted time. Discovering how information was classified, and presented, can make our reading of the book more expedient. A first exposure to data sets is much like the first contact with a new book. A careful inspection of the structure of the data is good research practice. Researchers should develop the habit of scanning the data prior to any conceptualization, or manipulation. In addition, we have also found it fruitful to try to visualize what the data will look like once it is "cleaned." Such an exercises can save a lot of the researcher's time down the line; making the process of manipulating the data a more efficient one.

This is also the time when the researcher's knowledge about data management, and his/her creativity should be put to work. However, developing a feel for what our final product will look like is not always an obvious task, especially when dealing with computer programs that we know little about. Even so, such steps should be pursued despite apparent barriers.

As a result of the process of visualizing the configuration of our final product, the research team decided to aggregate the counties so that in the end we would have streams of migrants across all the states in the country and between the 3088 counties. In addition, it was also decided that the format of

the data should be maintained, including the size of individual records. The final product than, would generate two migration files - for 1965-70 and 1975-80 - which could be used in connection with existing files for the 3088 counties.

Note, however, that the research question for this project calls for the analysis of the effect of migration to non-metropolitan counties. The research team could have chosen to analyze only these counties, which have previously been identified by the USDA. According to the USDA classification there are 242 non-metropolitan, chronically poor counties in the United States. Note further, that if such county classification was to be employed there would be a considerable reduction in the number of records, which could in turn translate into further savings in machine time, and storage space. In addition, if appropriate to the research questions, the variable age, which originally was divided into 17 intervals, could have also been easily reduced yielding greater savings. It should be kept in mind, that the exploration and implementation of these alternatives is contingent upon the types of questions posed by the project. We realize that data savings, when inappropriate or incorrectly done, can reduce the research's explanatory power. Keeping that fact in mind, consideration of one's end product, and seeking data reduction is encouraged at all times.

IMPLEMENTING COUNTY AGGREGATION

Upon the identification of our final product, we were than ready to start with the execution of our task. We chose not to create our own database. The demo database is available to all Ingres users, and by using demo, disk space could be saved. However, creating your own database has its advantages, especially if more than one person will be working, and updating the data.

There are a number of procedures that one should became familiar with whenever using demo. Specifically, 'restoredb' and 'purgedb' are very useful commands that will restore the integrity of the database, and purge unwanted extraneous files. We encourage readers to seek a more complete description of these two commands by referring to the Ingres manual.

Next, a series of extract runs were executed, so that the data could be copied into disk and later copied into an Ingres table. For a more detailed explanation of extract utilities the reader should refer to CDE documentation. For our particular task, however, these extract utilities were used to select, and to copy, data from individual states onto disk. On a number of occasions two to three states were copied onto disk at once. However, the number of states to be copied onto disk often depended upon the number of records for individual states, as well as to the extent that the efficiency of the system was being compromised.

Following, a number of Ingres routines were developed. These routines consisted of a collection of statements that would create a table in the demo database, copy the data from disk into the table, sort the data, aggregate the counties, and finally copying the data back onto disk for later storage onto tape.

The generic form of this routine took the following form:

```
i cdepublic:bench.ing |g
bench |g
create trial (dfips=i4, ofips=i4, age=i1, sex=i1, race=i1,
weight=i4) |g
bench |g
copy trial (dfips=c5, ofips=c5, age=c2, sex=c2, race=c1, weight=c0nl)
from "sys$userdisk:[00000]filename.dat" |g
bench |g
range of t is trial |g
replace t (dfips=xxwww) where t.dfips=zzyyr |g
replace t (dfips=xxwwt) where t.dfips=zzyys |g
bench |g
retrieve into checka (t.dfips, total=count(t.dfips by t.dfips))
```

```
where t.tid=min(t.tid by t.dfips) |g
bench |g
print check_a |g
destroy check_a |g
bench |g
replace t (ofips=xxwww) where t.ofips=zzyyr |g
replace t (ofips=xxwwt) where t.ofips=zzyys |g
bench |g
retrieve into checkb (t.ofips, total=count(t.ofips by t.ofips)) where
t.tid=min(t.tid by t.ofips) |g
bench |g
print check_b |g
destroy check_b |g
bench |g
copy trial (dfips=c5, ofips=c5, age=c2, sex=c1, race=c1, weight=c0)
into "sys$userdisk:[00000]filename.txt,text" |g
bench |g
q |g
```

The above sequence of statements is not as complicated as it might appear at first glance. However, we would like to give a more detail explanation on these commands, and call your attention to a number of possible problem spots.

Figure 2 illustrates the configuration of a Ingres table according to specifications in the 'create command.' Storage formats may vary according to the type of the data, and the range of variables to be stored. For this exercise the integer storage type was selected, however other storage formats may be chosen. The reader is encouraged to consult the Ingres manual for the selection of particular storage formats, as well as for the specifics on table names, "ownership", and size limitations.

```
|dfips     |ofips     |age  |sex  |race |weight     |
```

Figure 2: The Create Command

We have called your attention to the importance of familiarizing yourself with the structure of your data set, and trying to visualize your final product. Despite such familiarity, it is recommended that before copying the data into an

Ingres table that a 'copydb' be executed first. This is not the place to expand on the technicalities of 'copydb.' Suffice to say that such command will indicate to you the specifications for copying your data in and out of database in an efficient, error-free manner.

Our next step was to delineate the range of our table by defining the letter "t" to be equal to the table name. This simple command is a saving device. It allows one to refer to the table by simply typing the range designation, for our case "t".

Next, comes a series of replace commands. These commands constitute the heart of our task. Through them we were able to assign the correct 'dfips,' and 'ofips,' to the respective streams, without changing the value of the other variables. Table 1 represents a section of the table 'Trial' prior to the execution of the replace statements.

Table 1: Selected Streams from Table Trial Prior to Dfips and Ofips Aggregation

dfips	ofips	age	sex	race	weight
01001	01003	07	1	2	0000000009
01001	01003	07	2	2	0000000002
01001	01003	09	2	1	0000000005
01005	01007	02	1	1	0000000005
01005	01007	02	2	2	0000000004
01005	01007	03	1	2	0000000002
01005	01007	09	2	1	0000000021
01007	01003	08	1	1	0000000019
01007	01003	08	2	1	0000000019
01011	01005	04	1	1	0000000003
01011	01005	05	2	2	0000000010
01011	01027	05	2	1	0000000009
01011	01027	06	1	2	0000000015

The following replace commands were then executed:

```
replace t (dfips=01009) where t.dfips=01005 |g
replace t (dfips=01015) where t.dfips=01007 |g
replace t (ofips=01009) where t.ofips=01005 |g
```

```
replace t (ofips=01015) where t.ofips=01007 |g
```

resulting in the following table:

Table 2: Selected Streams from Table Trial After Dfips and Ofips aggregation

dfips	ofips	age	sex	race	weight
01001	01003	07	1	2	0000000009
01001	01003	07	2	2	0000000002
01001	01003	09	2	1	0000000005
01009	01015	02	1	1	0000000005
01009	01015	02	2	2	0000000004
01009	01015	03	1	2	0000000002
01009	01015	09	2	1	0000000021
01015	01003	08	1	1	0000000019
01015	01003	08	2	1	0000000019
01011	01009	04	1	1	0000000003
01011	01009	05	2	2	0000000010
01011	01027	05	2	1	0000000009
01011	01027	06	1	2	0000000015

A comparison between table 1 and table 2 indicates that only those county codes whose values were 005 and 007 changed, while the values for other variables remained the same.

The reader has perhaps also noted the frequency of 'bench' statements in the above Ingres routine. Bench's are useful commands that will allow you to have a relatively precise estimation of necessary cpu time for the execution of your commands. By including such statements in our Ingres routine we were able to estimate the number of replacement statements that a single run could execute given the cpu time limits.

Lastly, we would like to suggest a procedure through which one can verify whether or not the data has been aggregated. The commands 'retrieve into check_a,' 'and retrieve into check_b' were implemented with this in mind. Specifically, table check_a was created by retrieving 'difps' codes and counting

the number of times any given 'dfips' appear on the table. Table check_a is then printed onto a log file where the replace commands can be verified. Note also that Ingres allows for a number of 'where clauses' with various specifications. This flexibility is sufficient to fulfill most conditions. Table 3 provides a visual description of table check_a.

Table 3: check_a

dfips	count
01001	4
01009	4
01015	2
01011	4

A similar table, i.e., check_b, was implemented to verify whether 'ofips' was aggregated according to the 'replace command' specifications.

OPTIMAL ORGANIZATION: A PRIORI ESTIMATES OF REDUCTIONS IN MIGRATION FILE SIZE

The first part of this paper was devoted to describing how a particular "data processing problem" was solved and how an efficient methodology was developed to support that process. At this point an alternative data structure is described that may also be used to solve the "data processing problem". The solution described here was modeled and tested with INGRES with data reorganization as the first step. The task could also be accomplished by resorting to computer programming or other software packages. However INGRES query languages, SQL or QUEL, happen to lend themselves to the task by being compact and concise. QUEL been used for the process of modeling the data reorganization served as the starting point to solve the aforementioned data processing problem with considerable savings in processing costs; elapsed clock

on the wall time too! This "new" structure facilitates not only solving the case study problem as cited but is a superior data structure from which to launch almost any study of migration using the Census files described in the early part of this report.

As to file size estimates, first consider that each of two the files consists of approximately 2.4 million records, where the basic record is 24 bytes and is composed of migration flows between all counties disaggregated by sex, race and age. Using the approximate number, 3000, as representing the number of counties, with two sex categories, two race categories and 17 age intervals then several interesting numbers can be estimated. These estimates can be used to develop an idea of how much space savings can be derived from restructuring the data. If flows did occur between all counties there are a potential of 3,000 * 3,000 flows. The number 3,000 has been used to keep the calculations simple. The potential number of flows, 9,000,000 ignores the impact of disaggregation by sex, race and age (2 * 2 * 17). Thus, given the current data structure, the potential total of flows is on the order of 612,000,000. Given that each record takes up 24 bytes of storage, if all these flows existed this would amount to 14.7 gigabytes of data (one gigabyte stands for 1,000,000,000 bytes of data).

In reality roughly 2.4 million flows (one for each disaggregated category) actually occur. These 2.4 million flows require 57,600,000 bytes of storage (24 * 2.4 million; 57.6 megabytes of data) or only .39 percent of the potential of 14.7 gigabytes.

As a first step toward identifying the alternative data structure consider placing all values associated with the various sex, race and age categories on a single record along with the state-county of origin (ofips) and state- county of destination (dfips). This structure can easily be processed to regain the

original representation and the break even point on size is roughly 60,000 unique ofips and dfips pairs. There will be a record where any ofips and dips pair has at least one flow out of the 68. Note that the 10 bytes used to represent the ofips and dfips are no longer repeated up to 68 times but only once for each pair (this represents 23.4 megabytes). By dividing the number of records, 2.4 million, by the number of counties, 3,000, the number of flows per county is estimated, quite roughly, as 800. How many of these might be different categories of flow involving the same counties? If 800 is divided by 68 then, again speaking roughly, on the average there are 12 flows per group. As a reasonable working assumption assume on the average that each county has flows to 20 (20*3000) other counties and thus one breaks even on the space. This is only an estimate, the real answer depends on the number of flows that actually do occur.

Just as important to space savings is the fact that it is no longer necessary to explicitly represent the sex, race and age interval along with the value of the flow. The value stands by itself identified by position. By eliminating the space for sex, race and age an additional 4 bytes per category has been saved (4 * 68 * 60000;16.3 megabytes). The figure below suggests how the data are now represented.

[dfips: ofips: 1st flow: ... : nth flow]

Figure 3. Data structure for count flow migration data.

This record contains the dfips and ofips and the flows for each of the 68 combinations of sex, race and age. At this point it is fairly safe to estimate that the data now requires only 41 megabytes; given the earlier assumptions (57 - 16). Each record takes up 690 bytes but now there are many fewer than 2.4 million; with an acceptably conservative break even space assumption of 60,000

unique dfips and ofips pairs.

CHOICES OF DATA REPRESENTATION AND COMPRESSION

At this point it becomes necessary to consider facts relating to the physical representation of data in machine readable formats, computations with those different formats and the space that each of the formats requires. All of the estimates generated so far deal with the data in ASCII format. This is generally the bulkiest of representations and the least convenient for performing calculations. It has the advantage of being directly decipherable, by text editors and such, and with being portable from computer to computer. Examples of names for other representations usually indicate the storage requirements, in bytes, not the range of values that can be represented by the choice of format. For example, INTEGER1 format means that it takes one byte of storage; this one byte can however represent values between 127 or -128. INTEGER2 requires 2 bytes and can represent values from 32767 to -32768. INTEGER4 requires 4 bytes and can represent values from 2,147,483,647 to -2,147,483,648. FLOAT4 requires 4 bytes and can represent values from 10**38 to -10**38 but can only maintain 7 digits of precision; and after manipulation perhaps considerably less precision in reality. All these remarks pertain to VAX computers. There are yet other representations but their description is not required for this discussion.

An additional but perhaps advantageous approach, if data is left in ASCII, relies on file compression techniques. Since many of the flows are zero or blank they could be eliminated from the record but counted and put back when the record is actually used. This may save even more space than the approach already described but there are tradeoffs generally requiring more computing resources to process the data. Evaluation of this technique is delayed for

another paper until the advantages of alternative data representations, described below, is understood.

By using just the alternative data representations the storage requirements for the migration data can be reduced even more. Recall that the ASCII record is 690 bytes, where each of dfips/ofips takes 5 bytes and each of the flow values requires 10 bytes (68 of them). It can be seen from the above descriptions that INTEGER4 can be used to represent the values of the flows (perhaps INTEGER2 could be used). This saves 6 bytes per disaggregated flow for a total of 408 bytes. The 690 byte record is now 282 bytes. Some further savings can be achieved in the dfips and o codes but have been ignored for the purposes of these estimates. Using the earlier estimate of residual file size (41 megabytes) it can be estimated that the total size is now no more than 17 megabytes (60000 * 282) and probably quite a bit smaller if there are less than 60,000 flows; and could be made to be even smaller by using INTEGER2 where appropriate.

RESTRUCTURING THE MIGRATION FILES

There are a multitude of ways to proceed to transform the migration data from its approximate 612 million flows and yet avoid the full 14.7 gigabyte "sparse" matrix. All the approaches revolve around sorting or rely on data ordering. The easiest approach is to use a relational database system like INGRES to do the work combined with a few other processing packages. Consider the following broad outline of the process:

(1) Within an INGRES database create an empty BTREE table, call it CNTYFLOW, with 70 fields: dfips, ofips, and 68 flow fields. The BTREE should by keyed unique on dfips and ofips. Make sure the data representations are optimal.

(2) Within an INGRES database create an empty BTREE table, call it FLOW, with 3 fields: dfips, ofips and weight. Key the table on dfips and ofips. Make sure the data representations are optimal.

(3) Construct a runstream (using a macro) which will pass over the FLOW data moving data from the FLOW table to the CNTYFLOW table. Each pass is for a different category of age, sex and race and moves the weight to a corresponding weight field in CNTYFLOW (the macro accomplishes this). This process will be used 68 times.

(4) Use a package such as ODH and extract all records for a particular age, race and sex category to a disk file. Only dfips, oips and the flow are necessary in the disk file.

(5) Using an INGRES copy statement move these records into the empty FLOW table. Delete the original disk file.

(6) Append all unique combinations of dfips and ofips that occur within the contents of FLOW to the table CNTYFLOW. This will be a number far less than the number of records loaded. INGRES will do this by an APPEND statement and by the fact that CNTYFLOW was created keyed uniquely on dfips and ofips as a BTREE. An intermediate table may be used to speed up this step or the CNTYFLOW table could be constructed completely in a separate step.

(7) Use the module constructed in step (3) to replace the weights in CNTYFLOW. Modify the FLOW table to truncated. Modify to BTREE keyed on dfips and ofips.

(8) Repeat steps (4) thru (7) until data is exhausted (68 steps).

Some of the 68 steps will be very short and involve very little data. The short pass will take minutes and the longest pass less the 15 minutes, with most of the time involved in passing the data on tape.

TEST RESULTS

The process was tested so that the estimates given earlier could be refined. A full set of records for all counties in the U.S. was extracted for sex = 2 (female), race = 2 (nonwhite) and age = 16 (80 to 84 years old). The resulting disk file required (568*512) megabytes. Only the dfips, ofips, and flow were extracted. In Figure 4 the raw data is copied into an INGRES database table called FLOW. An extra step was added copying FLOW to FLOWSUM (the statement beginning with the word retrieve). This table is not necessary but was added to check out the situation with duplicates that had been identified in an earlier test. The last important step is to take advantage of the properties of a UNIQUE BTREE to prevent records with duplicate keys from being added to the table. In this case it was expected originally that all 13208 records would be added; instead there were 24 duplicates. With each successive pass of another of the 68 categories fewer and fewer pairs would be added by this process. This process took just under 15 minutes of cpu and 50,000 input/output disk requests. The additional table copy is negligible but it should be noted that the cost of will be linear in the number of rows to be considered for insertion or decreasing as the actual number of rows to be inserted decreases. With sufficient diskspace (roughly 60 megabytes) INGRES could be used to copy all 2.4 million cases of just dfips and ofips pairs and in one step eliminate the duplicates. In terms of total computing resources external extract and sort routines using tape could probably do as well. One step processing with INGRES (which would essentially sort the data as well) would do just as well. Either would do better than making 68 passes just to identify unique pairs; but the purpose of this case study was to demonstrate the utility of the data structure and the feasibility of accomplishing the task in a highly constrained computing environment.

```
... beginning of Figure 4

modify flow to truncated

copy flow  (
            dfips       =c5,
            ofips       =c5,
            flow        =d9,
            nl          =d0nl ) from
                 "STAFFB$DISK:[003001.MIGRATE.data]flow.dat"
(13208 rows)
continue
Executing . . .

modify flowsum to truncated

continue
Executing . . .

/*
This is an additional step added to create an intermediate
table where the flows from duplicates could be summed
*/

append flowsum ( flow.all)

(13208 rows)

continue
Executing . . .

retrieve ( cpu = cpums, dio=diocnt )

|cpu          |dio          |
|-------------------------|
|       118580|         1286|
|-------------------------|
(1 row)
continue
Executing . . .

modify cntyflow to truncated

modify cntyflow to btree unique on dfips,ofips
(0 rows)

continue
Executing . . .
/*
This step picks out the unique pairs of dfips and ofips! This
step revealed that dfips and ofips were not unique within
```

```
sex, race and age categories as originally assumed
...started with 13208 and ended with 13184; since this was
the first insertion into a UNIQUE BTREE structure there must
have been 24 duplicate dfips-ofips pairs
*/

append cntyflow ( dfips = flow.dfips, ofips= flow.ofips )
(13184 rows)

continue
Executing . . .

retrieve (cpu=cpums, dio=diocnt )
|cpu          |dio          |
|-------------------------|
|       830580|        50444|
|-------------------------|
(1 row)

... end of Figure
```

Figure 4. Edited script of INGRES code to perform the first insertion of dfips and ofips pairs into CNTYFLOW.

In Figure 4 the table CNTYFLOW, to which new county flow records were added, is now processed to put the flows into place. The portion of the "replace" that does this is "w2216=flowsum". The actual replacement is controlled by the conditions expressed in the "where clause". The dfips and ofips have to match between the CNTYFLOW table and the FLOWSUM table. Two other columns in the table are updated as well. The "flowcnt" is incremented by one to reflect that another flow column in this column has been updated. The "flowsum" column is updated to reflect a running tally of the sum of all flows for this county.

```
... beginning of Figure 5

replace cntyflow (flowcnt=cntyflow.flowcnt + 1,
                  flowsum =
                   cntyflow.flowsum + flowsum.flow,
                  w2216=flowsum.flow )
          where cntyflow.dfips = flowsum.ofips
                and
```

```
                cntyflow.dfips = flowsum.dfips
(13184 rows)
continue
Executing . . .

retrieve (cpu=cpums, dio=diocnt )

|cpu          |dio          |
|-------------------------|
|       486820|        17555|
|-------------------------|
(1 row)

... end of figure
```

Figure 5. Edited script of INGRES code to perform the very first single pass to place flows into position locations, e.g., in this case W2216 (sex=2, race=2, agegroup=16).

Summarizing, Figure 4 is the code used to find and insert new pairs of dfips and ofips. An additional step had to be added to construct sums of flows for duplicate dfips and ofips. The process of creating a new table is shown in the figure but not that for creating the new sums and then removing the duplicate cases. The time and effort of creating the unique pairs of dfips and ofips could be approached differently and more efficiently with INGRES or with a "programmed" approach with equivalent results but much savings in cpu and direct input/output costs over the particular attack used for purposes of this illustration. Figure 5 is the code used to replace or update a particular flow, and was illustrated for W2216. Additional calculations were performed to provide the capability to cross check the process later. The same problems were executed on an IBM PC by moving data to the PC and then using INGRES on the PC to perform the task. The results, speaking very subjectively, were about the same; that is to say that the clock on the wall time for the two solutions was about the same. To make truly valid comparisons is not quite practical but leave it to say that the VAX 11/780 was heavily loaded and the speed of the PC's

hard disk drive is unknown.

NEW ESTIMATES

The question remains as to how much space the table will occupy when the process is completed for the other 67 flows. However, consider that there are 68 flows and thus one flow represents roughly 1.5% of the total group of flows. This group had roughly 13000 flows or (13000/2400000) 54% of the total number of flows represented on the original file. On the average for each dfips county (roughly 2500) there were 5.2 ofips counties and symmetrically for each dfips (roughly 2500) there were 5.2 origins. It is at least somewhat reasonable to conclude that the flow count represents only one-third of what one might expect. Multiplying 3 times 13000 provides an estimate of 39,000 flows per group. Another data point (a different sex, race, age category) would add confidence to the estimation process. The question remains just how much overlap in dfips and ofips pairs there will be from group to group. There is no reason to expect wide divergence of target counties so accepting 39,000 unique flows as a bottom line estimate one can derive a new estimate of table size. The record size is 282 bytes (ignoring extra fields that were added and additional savings that could be achieved from different data type selections) just multiply this times 39000 and one has a new estimate of 11 megabytes.

CONTRAST: CAST STUDY OF COUNTY RECODING

Now consider the problem described in the first part of this paper. Essentially two sets of state-county codes (dfips and ofips) need to be made consistent with one another. Since the data as now structured is so compact and can thus be processed quite reasonably with a VAX 11/780 or even a personal computer it is not necessary to consider restricting the data set (eliminating

or grouping flows.) All that is necessary is to modify the generic code presented earlier to now change the dfips and ofips in the CNTYFLOW table. It would be wise to make a backup of the original table first, since others are probably not necessarily interested in the recodes but would rather have the data as it is. Consider now that rather than potentially scanning 2.4 million records to see if they are candidates for having dfips or ofips changed one only needs to scan the flows for the 39,000 to 60,000+ records making only a small number of changes to the total number of records.

Mailing Address:

Center for Demography and Ecology
1180 Observatory Drive, #4412
University of Wisconsin
Madison, Wisconsin 53706
U.S.A.