RESEARCH REPORT SERIES
(*Statistics #2002-06)*

Developing SAS Software for Generating a Complete Set of Ratio Edits

Maria Garcia and Roger Goodwin

Statistical Research Division
U.S. Bureau of the Census
Washington D.C. 20233

Report Issued: November 8, 2002

Developing SAS® Software for Generating a Complete Set of Ratio Edits

Maria Garcia and Roger Goodwin

Abstract

Survey data editing using the Fellegi-Holt model requires the complete set of explicit and implicit edits. This report presents new SAS® software that generates the complete set of implicit (implied) edits from a given set of explicit ratio edits. We describe the corresponding methodology, along with detailed discussions on the considered alternatives. The new software implements a shortest path algorithm and borrows ideas from the Generate Edits portion (subroutine) currently used in the U.S. Census Bureau's Plain Vanilla Ratio Module.

Keywords: ratio edits, implied edits, shortest path

1. Introduction

Data items collected by the Economic Census programs are subjected to ratio edits as part of the overall data review process. A ratio edit compares the ratio of two highly correlated items to upper and lower bounds. Reported items that fall outside of the bounds (or tolerances) are considered edit failures, and one or both of the items in an edit-failing ratio are either imputed or flagged for analyst's review. For the 1997 Economic Census, the Census Bureau developed a generalized edit and imputation system, called Plain Vanilla (PV). The PV subsystem consists of three separate edit and imputation programs: a ratio edit module; a balance edit module; and a verification module. The ratio edit module is the "core" component of PV and is used to validate most programs' basic data items. The Ratio module utilizes the Fellegi-Holt model of editing which means that the complete set of ratio edits is tested simultaneously (Fellegi and Holt, 1976). The program determines the minimum number of fields to change so that the imputed edit-failing record satisfies all the edits (Greenberg, 1986). This methodology has been used successfully at the Census Bureau by other economic programs since the early 1980s (Greenberg and Petkunas, 1990; Winkler and Draper, 1997).

The Plain Vanilla Ratio Module consists of three main programs. The first program (Edit Generation) generates all implicit (implied)[1] edits from the user-supplied explicit edit set. The explicit edits imply other relationships between the data items that must be satisfied for the data record to satisfy the edits (any two ratio edits with a common item imply a third ratio edit). The set of explicit and implicit edits is called the complete set of edits. The PV Edit Generation program generates the complete set of edits from the user-supplied explicit edits, then checks the logical consistency of the edit set. The set of implicit edits obtained using the PV Edit

[1]Hereafter, the terms "implicit edit" and "implied edit" are used interchangeably.

Generation program are returned to the survey staff for evaluating the logical implications (implicit edits) of the explicit ratios and bounds. The other two programs in the PV Ratio module perform error localization and imputation and are not discussed further (they are outside the scope of this project).

Clearly, developing sets of ratio edits for the PV Ratio module is an iterative process. And prior to this project, it was unfortunately a cumbersome one. First, subject-matter experts (analysts) selected pairs of highly correlated items for explicit ratio tests. Then, they developed tolerances for each explicit ratio test in a given industry. After developing initial sets of explicit edit bounds in the subject-matter division, analysts transferred the data to an Economic Statistical Method and Programming Division (ESMPD) programmer and requested that s/he runs the Edit Generation program to obtain the complete set of ratio edits[2]. After doing so, the programmer supplied the implicit edits results to the analyst for review. If the analysts detected inconsistencies or unreasonable edit restrictions, they adjusted the "culprit" explicit edits and repeated the entire process until the subject-matter specialists were satisfied with the complete (explicit and implicit) set of ratio edits. This could take a while.

For the 1997 Economic Census, Thompson and Sigman (2000) developed SAS software that automatically generates explicit tolerance limits for ratio edits. The output from this software can be used (in its present form) as input to the PV Edit Generation software. However, to eliminate the "bottleneck" in the ratio-edit development process caused by having to examine the implied edits, the Statistical Research Division (SRD) and ESMPD decided to develop new SAS software that generates the complete set of edits from a given set of explicit ratio edits. In doing this, we decided to revisit the original Operations Research methodology used in the existing FORTRAN programs and to consider an alternative methodology based on shortest path algorithms. This alternate approach was presented in an invited lecture given at the Census Bureau by B. Greenberg on an idea suggested by J. Fagan (Fagan, 1999). The specific objectives of this project were:

- To produce a quality SAS software product that
  - uses a thoroughly tested method of finding the set of all implicit edits from any given set of explicit edits;
  - can be easily maintained by good SAS programmers; and
  - can be used in conjunction with existing ratio edit parameter developing programs and ratio edit programs (e.g., Plain Vanilla Ratio Module).
- To provide a simple-to-use User's Guide for this software.

---

[2]The PV Edit Generation software are FORTRAN programs, which (in the Economic Directorate) are maintained by ESMPD programmers.

This report presents the software developed for this project, along with specific details of the software development decision. Section 2 describes the methodology and available algorithms. Section 3 describes the Edit Generation software. Section 4 presents the results of the software testing. We close with a discussion in Section 5.

2. Implicit Edit Generation

2.1 Methodology

A ratio edit requires that the ratio of two data items is bounded by lower and upper bounds (of the form $l_{ij} \le v_i/v_j \le u_{ij}$, where l_{ij} and u_{ij} are the lower and upper bounds respectively). Any pair of ratio edits that contain a common data item implies another ratio edit: for example, the two ratio edits $l_{ik} \le v_i/v_k \le u_{ik}$ and $l_{kj} \le v_k/v_j \le u_{kj}$ imply the ratio edit $l_{ik}l_{kj} \le \frac{v_i}{v_j} \le u_{ik}u_{kj}$.

In what follows, we refer only to upper bounds. (Note that the ratio edit $l_{ij} \le \frac{v_i}{v_j} \le u_{ij}$ is equivalent to the following two edits: $\frac{v_i}{v_j} \le u_{ij}$ and $\frac{v_j}{v_i} \le 1/l_{ij} = u_{ji}$.) We can use graph theory to generate the complete set of edits. First, the explicit ratio edits are associated to a directed graph with n nodes $v_1, v_2, \ldots, v_n$ (one per unique data item field). Each pair of nodes is connected by a directed arc (v_i, v_j) associated to upper bound u_{ij} if there is an edit connecting fields v_i and v_j. To find the upper bound of the ratio edit connecting fields v_i and v_j, we search along the edges of the graph, traversing the graph by starting at node v_i and finishing at node v_j. As we traverse the graph, we compute the upper bound of the implied edit, always choosing the bounds that yield the optimal (smallest) upper bound. At every step, we retain the upper bounds contributing to this optimal bound. For example, assume that fields v_1, v_2, v_3, v_4 are restricted by the following upper bounds,

$$v_1 / v_2 \le 2$$
$$v_2 / v_3 \le 1$$
$$v_2 / v_4 \le 6$$
$$v_3 / v_4 \le 4$$

These edits are associated with the graph displayed in Figure 1.

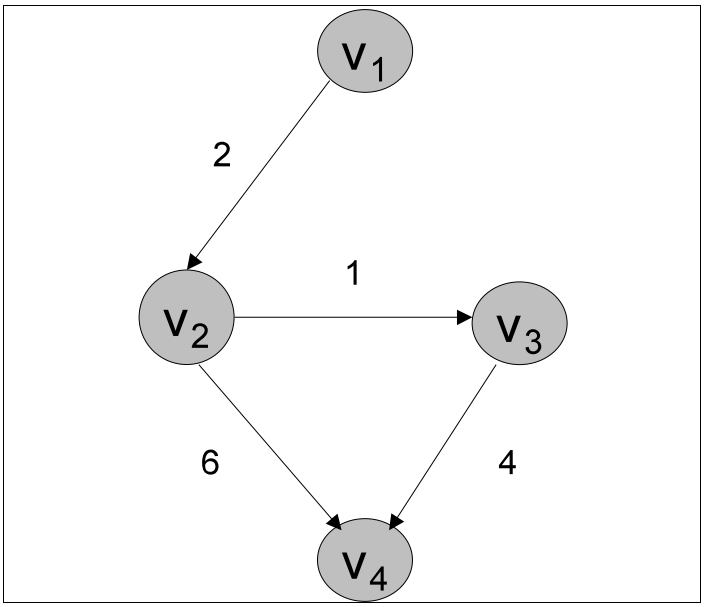


Figure 1: Graph associated with explicit edits

Items v_1 and v_4 are both tested against item v_2, creating an implied edit between those two items. To find the upper bounds for this implied edit, we traverse the graph from v_1 to v_4 and choose the path that will yield the optimal (minimum) upper bound. There are two possible ways of traversing the graph starting at node v_1 and ending at node v_4 :

1. traverse arcs $(v_1, v_2), (v_2, v_4)$, or
2. traverse arcs $(v_1, v_2), (v_2, v_3), (v_3, v_4)$

Following the path described by (1) yields an upper bound of $u_{14} = u_{12}u_{24} = 12$. The path described by (2) gives an upper bound of $u_{14} = u_{12}u_{23}u_{34} = 8$. Therefore the optimal upper bound is obtained traversing the graph from v_1 to v_4 through the arcs $(v_1, v_2), (v_2, v_3)$, (v_3, v_4), even though this path contains more arcs (nodes).

James Fagan (Fagan, 1999) framed the problem of finding the complete set of ratio edits as a shortest path problem. The setup is the same but every directed arc (v_i, v_j) is assigned a length as follows,

$c_{ij} = \ln u_{ij}$ and $c_{ji} = \ln u_{ji} = -\ln l_{ij}$, if there is at least one edit involving fields *i,j*

$c_{ij} = \infty$, otherwise.

We now use a shortest path algorithm to generate the implied edits. With this set-up, given the shortest path from v_i to v_j of cost c_{ij} , the bounds for the ratio edit between fields

v_i and v_j are,

$$u_{ij} = e^{c_{ij}} \quad \text{and} \quad l_{ij} = e^{-c_{ji}} \quad .$$

In this framework, the shortest path from v_1 to v_4 in the previous example is (v_1, v_2), (v_2, v_3), (v_3, v_4), with length $\ln 2 + \ln 1 + \ln 4 = \ln 8$ and the optimal upper bound is $u_{14} = e^{\ln 8} = 8$ as before.

2.2 Available Algorithms

If the explicit set of ratio edits is consistent, then the methodologies presented in Section 2.1 will determine the optimal bounds for the implied ratio edits. We began by examining the first approach described in Section 2.1, i.e., traversing the graph and choosing optimal bounds at each node. This algorithm is similar to the one currently used by the FORTRAN edit generation program maintained in ESMPD. The FORTRAN program uses connected sets to simulate the graph corresponding to the edits. The SAS program that we wrote to evaluate this approach has some key differences from the FORTRAN program. First, we used different data structures to represent the graph and nodes. Second, we formulate the program as a linear algebra program, coding in SAS/IML: the SAS matrix language easily represents the data fields and edit bounds using vectors and matrices. Because the implementation of this algorithm requires similar steps as the implementation of Floyd's algorithm described below, we call it the Floyd-like algorithm.

The second approach described in Section 2.1 generates the optimal ratio edit bounds using a shortest path algorithm. We first evaluated PROC NETFLOW, a SAS/OR procedure. SAS PROC NETFLOW implements the shortest path algorithm using the Simplex Method and the Interior Point Algorithm (SAS Institute, 1999). Using this canned procedure is quite appealing from an implementational perspective. However, in our initial test runs, running PROC NETFLOW proved to be quite time-consuming, hence our decision to write our own SAS code for solving the shortest path problem. We considered two separate shortest path algorithms: Dijkstra's algorithm for finding all shortest paths (known as Modified Dijkstra's algorithm, Brassard and Bratley, 1988) and Floyd's algorithm (Brassard and Bratley, 1988). Of these two algorithms, we chose to program only Floyd's algorithm for reasons outlined in Appendix 2.

2.2.1. SAS/IML Applications

Floyd's Algorithm

Step 1: Begin:

Let n be the number of nodes. Construct an $n \times n$ matrix C, such that for each i, j, $c_{ij} = \ln u_{ij}$ and $c_{ji} = \ln u_{ji} = -\ln l_{ij}$ where u_{ij} and l_{ij}, are the

bounds for the edit involving fields v_i and v_j. If there is no edit involving fields v_i and v_j then set $c_{ij} = tol$, where tol is a user-specified value for infinity

Step 2: Determine the optimal bounds

For i = 1, 2, ..., n **do**
For j = 1,2 , ..., n **do**
For k = 1, 2, ..., n **do**
$C[i,j] \leftarrow \mathbf{min}(C[i,j], C[i,k] + C[k,j])$

Step 3: Compute the lower and upper bounds

For i = 1,2, 3, ..., n -1 **do**
For $j = i$+1, ..., n **do**
$U[i,j] \leftarrow \exp(C[i,j])$
$U[j,i] \leftarrow \exp(-C[j,i])$
return

Floyd-like Algorithm

Step 1: Begin:

Let n be the number of nodes. Construct an $n \times n$ matrix U, such that for each i, j, u_{ij} is the upper bound of the edit $\frac{v_i}{v_j}$ and u_{ji} is $\frac{1}{l_{ij}}$ (upper bound of the edit $\frac{v_j}{v_i}$). If there is no edit between fields v_i and v_j then $u_{ij} = tol$, where tol is a user-specified value for infinity.

Step 2: Determine the optimal bounds

For i = 1, 2, ..., n **do**
For j = 1,2 , ..., n **do**
For k = 1, 2, ..., n **do**
$U[i,j] \leftarrow \mathbf{min}(U[i,j], U[i,k] * U[k,j])$

Step 3: Compute the lower bounds

For i = 2, 3, ..., n **do**
For j = 1, 2, ..., $i-1$ **do**
$U[i,j] \leftarrow 1 / U[i,j]$
return

Upon termination of the above algorithms, the matrix U contains both the lower and upper bounds of the ratio edits. Both Floyd's algorithm and the Floyd-like algorithm terminate in a time of order $O(n^3)$.

2.2.2. SAS/OR Application (PROC NETFLOW)

PROC NETFLOW solves the shortest path problem by solving a linear programming (LP) problem using the Simplex Algorithm or the Interior Point Algorithm (see SAS/OR User's Manual, SAS Institute, 1999). These algorithms are described below.

The Simplex Algorithm:

Step 1: Initialization: Introduce slack variables. (Slack variables are the equivalent of equality constraints to the inequality constraints).
Step 2: Iteration: Determine the entering basic variable and leaving basic variable. (Non-basic variables are set to zero, the others are called basic variables). Determine the new basic feasible solution.
Step 3: Optimality Test: Determine whether the solution in (2) is optimal.

The Interior Point Algorithm:

Step 1: "Shoot" through the interior of the feasible region toward an optimal solution.
Step 2: Move in a direction that improves the objective function value at the fastest possible rate.
Step 3: Transform the feasible region to place the current trial solution near its center, thereby enabling a large improvement when implementing Step 2.

3. Implicit Ratio Edit Generation: SAS software

We wrote three separate programs to research the alternative approaches for generating implied edits discussed in Section 2. All three programs have the same main driver routine written in base SAS with a call to either SAS/IML (testing the Floyd and Floyd-like algorithms) or to SAS/OR (testing the Simplex Algorithm/Interior Point Algorithm using PROC NETFLOW). Each program imposes an indexing scheme on the data fields in the edit set, checks the edit set for redundancies, determines if the edit set is consistent, and finally generate*s* the implicit ratio edits.

All three programs require an input SAS (version 8.2) data set of explicit edits, containing at least four data items (variables): **CLASS**, the classification variable code[3] (e.g., NAICS code); **E_RATIO**, the explicit ratio test; **L_FENCE**, the ratio test's lower bound; and **U_FENCE**, the ratio test's upper bound. This data set **can** be created using the software described in Thompson and Sigman (1996). Each program produces one output file, containing the complete set of edits (explicit and implied) for each industry in the input data set.

To help identify redundant edits, we index the tested data items based on alphabetical

[3]Hereafter referred to as "industry"

order: for example, the ratio test for variables QPR3 and TOT1, QPR3/TOT1, is associated with new variables and new ratio edit v_1/v_2. The variable indices used in our programs should not be used in subsequent PV Ratio module applications. The indexing scheme used in the PV Ratio edit, is **directly** associated with item imputation order.

The following sections describe how the software identifies and deletes redundancies from the original explicit set of edits (Section 3.1), how the software determines whether this modified set of edits is consistent (Section 3.2), and how each program treats disconnected graphs (Section 3.3). Section 3.4 presents the different analytic outputs produced by each program and discusses the advantages/disadvantages of each set of outputs.

3.1 Determining a Set of Non-Redundant Edits

A ratio edit for any two data fields is redundant if it contains another ratio edit for the same data fields; that is, the redundant edit adds no new restrictions. Redundant edits must be removed prior to searching for shortest paths. In the framework of graph theory, the presence of redundancy indicates there are more than two arcs connecting some nodes, and consequently, the shortest path algorithm cannot be applied.

Our software eliminates the redundant edits by choosing the most restrictive edits for any two fields. Redundant edits are handled as follows: if the edits for fields i, k are

$l_{ik} \le \frac{v_i}{v_k} \le u_{ik}$ and $\hat{l}_{ik} \le \frac{v_i}{v_k} \le \hat{u}_{ik}$, then the non-redundant edit is

$\max(l_{ik}, \hat{l}_{ik}) \le \frac{v_i}{v_k} \le \min(u_{ik}, \hat{u}_{ik})$. For example, given the following two explicit edits

$$2 \le \frac{v_1}{v_2} \le 7$$

$$4 \le \frac{v_1}{v_2} \le 9$$

the new non-redundant explicit ratio edit is $4 \le \frac{v_1}{v_2} \le 7$.

3.2 Determining a Consistent Set of Ratio Edits

Once a set of non-redundant explicit edits is available, the software determines whether it is consistent. A set of ratio edits is consistent if none of the edits contradict each other, i.e, for every two fields v_i and v_j in a ratio edit $l_{ik} \le \frac{v_i}{v_k} \le u_{ik}$, the ratio is consistent if $l_{ij} \ge 0$, $u_{ij} \le \infty$, and $l_{ij} \le u_{ij}$. In the framework of graph theory, inconsistency causes negative cycles in the graph. Any algorithm that finds a shortest path will always keep choosing this negative cycle until termination.

Any inconsistent set of edits must be returned to the survey analysts for review. The

software will do a preliminary check for inconsistencies in the explicit edits only after the redundant edits have been removed. This sequencing is very important, since there may be hidden inconsistencies that become obvious when redundancies are removed. For example, given the following explicit edits

$$4 \le \frac{v_1}{v_2} \le 7$$

$$2 \le \frac{v_1}{v_2} \le 3$$

the software produces a non-redundant edit of $4 \le \frac{v_1}{v_2} \le 3$. This relationship is impossible to satisfy, and so, the set of edits is inconsistent.

If there are any inconsistencies in the user-supplied edits, then the software stops processing for that industry, prints a message to the SAS log window, and continues with the next industry. The revised (consistent, non-redundant) set of explicit edits is then used to generate the implicit ratio edits.

3.3 Disconnected Graphs

Disconnected graphs occur when the explicit ratio edits contain edits which have no fields in common with the other edits. For example, in the explicit edit set v_1/ v_2, v_2/ v_3, and v_4/ v_5, the third edit (v_4/ v_5) is disconnected from the other two edits (which are connected by data item v_2). When the explicit edit set contains disconnected edits, two (or more) distinct sets of nodes exist. Our programs appropriately handle this situation, each in a different way. The SAS/IML version connects these nodes with (0, ∞) bounds, as required by the PV Ratio Module. The SAS/OR program only produces the connected edits.

3.4 Audit Trails/Analytical Outputs

The SAS/IML and SAS/OR applications produce very different audit trails. Figure 2 displays a sample of the audit trail output produced by the SAS/IML applications (the output window has been edited to fit the screen capture). This "audit trail" prints the bounds as they are updated. Recovering the shortest path contributing to the optimal bounds requires the representation of the edits in a tree data structure which has prohibitive space requirements. The column headings v_i, v_j, and v_k, are used to represent the data fields and the column headings U_ik, U_kj, and U_ij are used to represent the ratio edits upper bounds. The first row of the audit trail output shows that the upper bound for the ratio REP_EMP/REP_QPR (3.7199987743) was obtained using the upper bounds for the ratios REP_EMP/REP_APR and REP_APR/REP_QPR, that is 0.525984356 × 7.0724513251 = 3.7199987743.

AUDIT TRAIL

v_i	v_k	U_ik	v_k	v_j	U_kj	v_i	v_j	U_ij
REP_EMP/REP_APR	0.5259843586	REP_APR/REP_QPR	7.0724513251	REP_EMP/REP_QPR	3.7199987743			
REP_EMP/REP_APR	0.5259843586	REP_APR/REP_SLS	2.7109327401	REP_EMP/REP_SLS	1.4259082186			
REP_QPR/REP_APR	0.3902042111	REP_APR/REP_EMP	200.9521091000	REP_QPR/REP_EMP	78.4123591930			
REP_QPR/REP_APR	0.3902042111	REP_APR/REP_SLS	2.7109327401	REP_QPR/REP_SLS	1.0578173711			
REP_SLS/REP_APR	33.4665717880	REP_APR/REP_EMP	200.9521091000	REP_SLS/REP_EMP	6725.1781851000			
REP_SLS/REP_APR	33.4665717880	REP_APR/REP_QPR	7.0724513251	REP_SLS/REP_QPR	236.6906999900			

Figure 2: GenBndsIml Audit Trail

The SAS/IML audit trail is not very informative. In contrast, the SAS/OR output automatically produces a very useful audit trail. With this output, the user can see (at a glance) which edits contributed to the lower and/or upper bounds of an implied edit. For each industry, GenBndsOR produces an audit trail listing the derivation of each implied edit.

Figure 3 displays a sample of the SAS/OR audit trail. For the edit REP_EMP/REP_QPR, the lower bound is calculated using the lower bounds from the two edits REP_APR/REP_QPR and REP_EMP/REP_APR, $2.5627607 \times 0.0049763 = 0.0127531$. Similarly, the optimal upper bound is calculated using the upper bounds from the two edits REP_APR/REP_QPR and REP_EMP/REP_APR, $7.0724513 \times 0.5259844 = 3.7199988$.

Explicit and Implied Edits for Industry XXX1

Obs	Lower Bound	Mnemonic Name	Upper Bound
1	1.9011972	REP_APR/REP_EMP	200.9521091
2	2.5627607	REP_APR/REP_QPR	7.0724513
3	0.0298806	REP_APR/REP_SLS	2.7109327
4	0.0127531	REP_EMP/REP_QPR	3.7199988
5	0.0001487	REP_EMP/REP_SLS	1.4259082
6	0.0042249	REP_QPR/REP_SLS	1.0578173

Audit Trail for Industry XXX1

Lower Bound	Mnemonic Name	Upper Bound	Lower Path Bounds	Edits Traversed	Upper Path Bounds
1.9011972	REP_APR/REP_EMP	200.9521091	1.9011972	REP_APR/REP_EMP	200.9521091
2.5627607	REP_APR/REP_QPR	7.0724513	2.5627607	REP_APR/REP_QPR	7.0724513
0.0298806	REP_APR/REP_SLS	2.7109327	0.0298806	REP_APR/REP_SLS	2.7109327
0.0127531	REP_EMP/REP_QPR	3.7199988	2.5627607	REP_APR/REP_QPR	7.0724513
			0.0049763	REP_EMP/REP_APR	0.5259844
0.0001487	REP_EMP/REP_SLS	1.4259082	0.0298806	REP_APR/REP_SLS	2.7109327
			0.0049763	REP_EMP/REP_APR	0.5259844
0.0042249	REP_QPR/REP_SLS	1.0578173	0.0298806	REP_APR/REP_SLS	2.7109327
			0.1413937	REP_QPR/REP_APR	0.3902042

Figure 3: Partial GenBndsOR output

In the SAS/OR audit trails, the Lower Bounds path or Upper Bounds path variable may be set to missing (denoted by " . "). In such cases, an edit contributes to either the lower or upper bound, but not both. Figure 4 presents such an example. Examining the implied ratio edit REP_QPR/REP_SLS, the edits REP_APR/REP_EMP and REP_EMP/REP_SLS contribute to the lower bound but do not contribute to the upper bound, REP_APR/REP_SLS contributes only to the upper bound, and REP_QPR/REP_APR contributes to both.

Explicit and Implied Edits for Industry XXX1

Obs	Lower Bound	Mnemonic Name	Upper Bound
1	1.9011972	REP_APR/REP_EMP	18.9765346
2	2.5627607	REP_APR/REP_QPR	7.0724513
3	0.2715995	REP_APR/REP_SLS	2.7109327
4	0.1350489	REP_EMP/REP_QPR	3.7199988
5	0.1428571	REP_EMP/REP_SLS	1.0000000
6	0.0384025	REP_QPR/REP_SLS	1.0578173

Audit Trail for Industry XXX1

Lower Bound	Mnemonic Name	Upper Bound	Lower Path Bounds	Edits Traversed	Upper Path Bounds
1.9011972	REP_APR/REP_EMP	18.9765346	1.9011972	REP_APR/REP_EMP	.
			.	REP_APR/REP_SLS	2.7109327
			.	REP_SLS/REP_EMP	7.0000021
2.5627607	REP_APR/REP_QPR	7.0724513	2.5627607	REP_APR/REP_QPR	7.0724513
0.2715995	REP_APR/REP_SLS	2.7109327	1.9011972	REP_APR/REP_EMP	.
			.	REP_APR/REP_SLS	2.7109327
			0.1428571	REP_EMP/REP_SLS	.
0.1350489	REP_EMP/REP_QPR	3.7199988	2.5627607	REP_APR/REP_QPR	7.0724513
			.	REP_EMP/REP_APR	0.5259844
			0.1428571	REP_EMP/REP_SLS	.
			0.3688767	REP_SLS/REP_APR	.
0.1428571	REP_EMP/REP_SLS	1.0000000	0.1428571	REP_EMP/REP_SLS	1.0000000
0.0384025	REP_QPR/REP_SLS	1.0578173	1.9011972	REP_APR/REP_EMP	.
			.	REP_APR/REP_SLS	2.7109327
			0.1428571	REP_EMP/REP_SLS	.
			0.1413937	REP_QPR/REP_APR	0.3902042

Figure 4: Partial GenBndsOR output with edits contributing to one bound but not the other

Clearly, the SAS/OR audit trail is the easier of the two to interpret and is the more useful for ratio edit development. This is a key distinction between the two approaches.

4. Software Testing Results

We used six separate test data sets to evaluate the performance of the three different implicit ratio edit generation programs. Five of these data sets were designed to test the specific situations listed in Table 1. The smallest of these five data set contains four items and one industry (10 observations) and the largest contains eight items and ten industries (80 observations). To assess the capability of the software of handling large data sets with several ratio edits, we also tested the programs on the Census Bureau's Annual Survey of Manufactures

(ASM) production edits (multiple industries; 7,095 explicit ratio edits). Because we did not evaluate the ASM data results (this was strictly a feasibility test), we do not list the ASM data in Table 1 or provide the associated run-times in Table 2.

Table 2 provides the result of the run-time study for six separate data sets. To avoid the confounding effect of network traffic, all run-times were obtained on a stand-alone laptop computer.

Table 1: Test Data Sets

Dataset	Number of Records	Number of Industries	Number of Explicit Ratios per Industry	Specifically Checks
Crazy	10	1	10	- explicit ratios that imply impossible relationship (inconsistent edit)
Wholesale	66	10	varies (min = 5, max=8)	- varying numbers of explicit ratios within industry in same input dataset (not all ratio edits are tested in every industry) - disconnected ratio test in three industries
Wholesale.c	80	10	8	- not applicable ratio tests have lower bound zero and upper bound infinity - disconnected ratio test in three industries
Retail	12	4	3	- mnemonic name of ratio tests differs by industry
Services	22	7	varies	- varying numbers of explicit ratios within industry in same input dataset (not all ratio edits are tested in every industry)
Services.c	28	7	4	- non applicable ratio tests have bounds lower zero and upper bound infinity

Table 2: Execution Time for Edit Generation Programs

Test Deck	Floyd-like Algorithm	Floyd's Algorithm	SAS Proc NETFLOW
Wholesale	10 secs	10 secs	73 secs
Wholesale.c	10 secs	9 secs	99 secs
Retail	5 secs	5 secs	24 secs
Services	7 secs	7 secs	22 sec
Services.c	7 secs	7 secs	29 secs
Crazy	4 secs	4 secs	4 secs

For each test deck, the run-time difference between the two SAS/IML programs (Floyd and Floyd-like algorithms) was negligible. However, both SAS/IML programs were

consistently faster than the SAS/OR (PROC NETFLOW) program. The SAS/IML code runs more quickly because it requires three simple DO loops, whereas PROC NETFLOW finds the shortest path (implied edits) by solving a linear programming problem using the computationally intensive simplex algorithm. The advantage of the SAS/OR program is the audit trail.

4. Discussion

In this report, we presented development of SAS software for generating the complete set of edits for a given set of explicit ratio edits. In Section 3, we presented the considered methodologies. The results presented in Section 4 show the strengths and weaknesses of our three separate programs. Both SAS/IML programs have similar running times. However, the SAS/IML code that implements the Floyd-like algorithm does not require the creation of the cost matrix. Creating the cost matrix for implementing Floyd's shortest path algorithm requires calculating the logarithm of the lower and upper ratio edit bounds. The main advantage of the Floyd-like implementation is that it is easier to explain and understand since it does not require this data transformation. Because there are no other differences between the two SAS/IML programs, we recommend using the SAS/IML program that implements the Floyd-like algorithm.

This SAS/IML code is not however without its disadvantages: the code generates an audit trail of edit updates but it does not provide a list of edits used to calculate the optimal bounds. Although the SAS/IML programs are considerably faster, the SAS/OR program provides more immediately useful outputs. The availability of the edits – explicit and implicit – contributing to the optimal bounds is a software feature that is quite useful to edit-developers and has been well received by the customers. Our recommendation is to use the faster, easier-to-maintain SAS/IML code implementing the Floyd-like algorithm, during production, when time is an issue, but to use the SAS/OR version implementing Floyd's shortest path algorithm during development, when a more detailed examination of the implied edits is needed. In a separate report (Goodwin, Garcia, and Thompson (2002)), we provide software documentation along with a detailed User's guide for running the programs.

5. References

1. Brassard, G., and Bratley P., "*Algorithmics: Theory and Practice*", Prentice Hall, NJ, 1988.

2. Fagan, J. , "Generating Implied Ratio Edits", unpublished Census manuscript. October 1999.

3. Fellegi, I. P. and Holt, D., " A Systematic Approach to Automatic Edit and Imputation," *The Journal of the American Statistical Association, No. 71*, 1976.

4. Goodwin, R., Garcia, M., and Thompson, K., "The GenBounds Software for Generating a Complete Set of Ratio Edits: %Implied User's Guide," SRD Research Report SS-Statistics # 2002-01.

5. Greenberg, B. and Petkunas, T., "SPEER (Structured Program For Economic Editing and Referrals", Proceedings of the Section on Survey Research Methods, ASA, 1990.

6. Hillier, Frederick S. and Lieberman, Gerald J, "*Introduction to Operations Research 5th Edition,*" McGraw-Hill Publishing Co, NY, NY, 1990.

7. SAS Institute Inc., *SAS/OR User's Guide: Mathematical Programming, Version 8*, Cary, NC: SAS Institute Inc., 1999. 566pp.

8. Thompson, K. J. and Sigman, R., "Statistical Methods for Developing Ratio Edit Tolerances for Economic Data", J. Official Statistics, 2000.

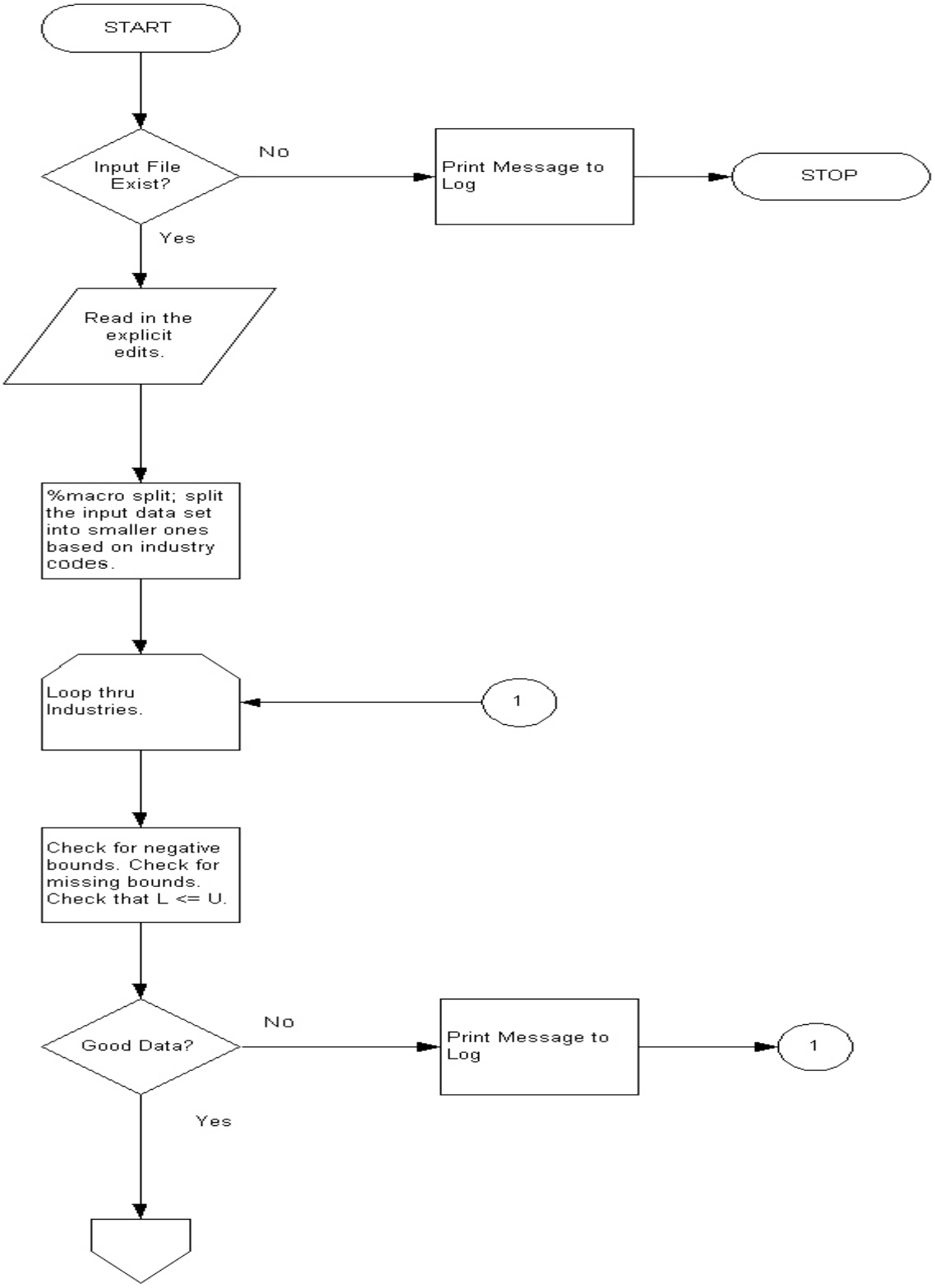
9. Thompson, K. J. and Sigman, R, "User Guide for Generalized SAS Ratio Edit Parameter Development Programs", Washington, DC.: U.S. Bureau of the Census (Technical Report #ESM-9602, available from the Economic Statistical Methods and Programming Division).

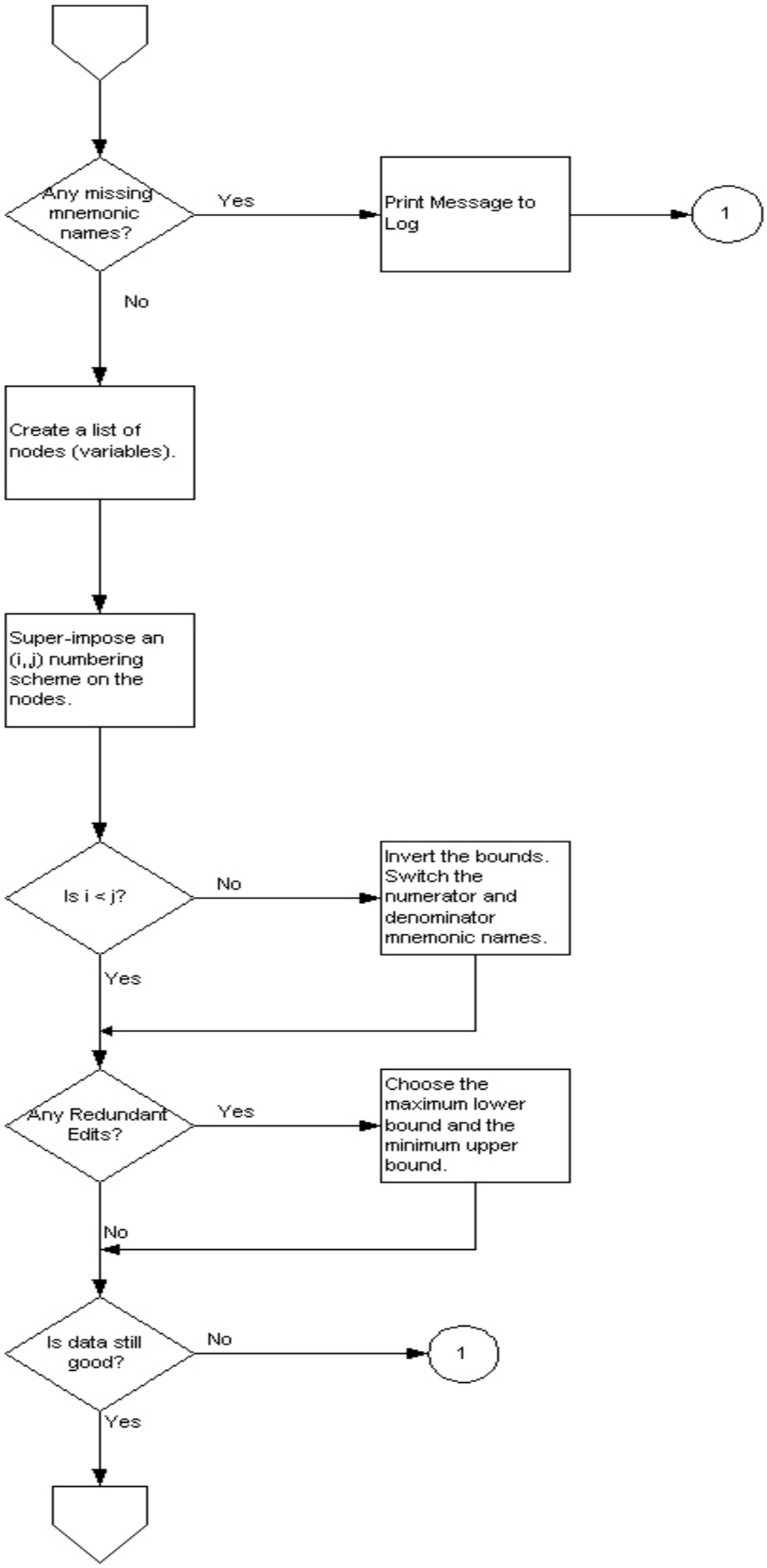
10. Winkler, W. and Draper, L. , "The SPEER Edit System", Proceedings of the Conference of European Statisticians, Section on Statistical Data Editing, UNECE, 1997.

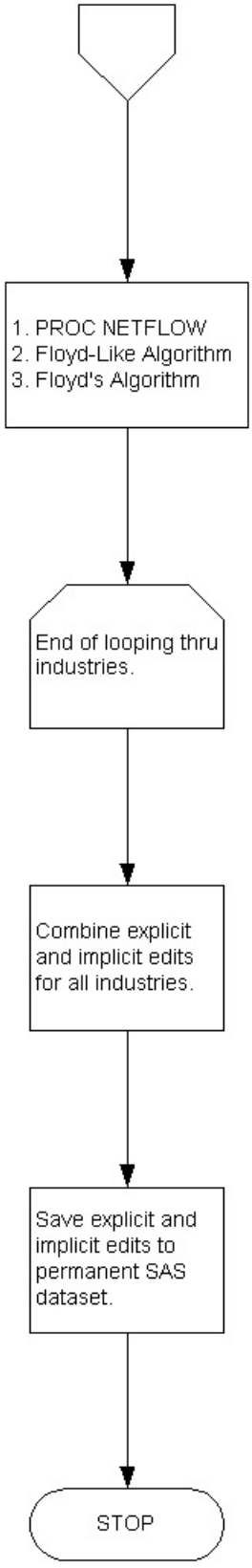
Acknowledgment

The authors would like to thank James Fagan for providing the new algorithm used in the GenBndsOR code, Brian Greenberg for his lecture and discussion on this methodology, and Richard Sigman for his contribution and helpful discussions on the SAS/OR code. The authors also wish to thank Robert Jewett for helpful discussion on producing an audit trail for the SAS/IML code. We also thank Katherine J. Thompson and William Winkler for providing advice and their helpful comments on earlier versions of this manuscript. Finally, we thank Katherine J. Thompson for her leadership in bringing about this project.

Appendix 1: Software Design Flowchart




Any missing mnemonic names?
Yes
Print Message to Log
1
No
Create a list of nodes (variables).
Super-impose an (i,j) numbering scheme on the nodes.
Is i < j?
No
Invert the bounds. Switch the numerator and denominator mnemonic names.
Yes
Any Redundant Edits?
Yes
Choose the maximum lower bound and the minimum upper bound.
No
Is data still good?
No
1
Yes


1. PROC NETFLOW
2. Floyd-Like Algorithm
3. Floyd's Algorithm
End of looping thru industries.
Combine explicit and implicit edits for all industries.
Save explicit and implicit edits to permanent SAS dataset.
STOP

Appendix 2: Investigation of Shortest Path Algorithms

We considered two different shortest path algorithms: Floyd's algorithm (described in Section 2.2.1) and the Modified Dijkstra's algorithm. The Modified Dijkstra's algorithms begins with a candidate set of nodes C, each of which is associated with elements in a cost array D, and uses a greedy algorithm to determine the minimum path associated with these costs. Upon termination, the set *S* of source nodes contains all nodes in the graph, array *D* contains the minimum costs, and array *P* contains the shortest path.

As described in Section 2, the implied edits are associated to a directed graph. The shortest path algorithm finds the optimal bounds for the implied edit between any two fields. Consequently, to find all optimal bounds one must solve *n* shortest path problems, where *n* is the number of fields. When given a starting node and an ending node, the shortest path for a given edit can be found in $O(n^2)$ iterations. However, finding **all** implicit edits requires $nO(n^2) = O(n^3)$ iterations.

The table below compares the maximum number of iterations for varying numbers of arcs and nodes using Floyd's and Dijkstra's shortest path algorithms.

Maximum Number of Iterations in Floyd's Algorithm vs Dijkstra's Algorithm

	a	n	Floyd's Iterations	Modifed Dijkstra Iterations
	5	5	125	35
	10	5	125	52
	10	10	1000	200
a = number of edges (arcs)	20	10	1000	300
	50	10	1000	600
n = number of nodes (items)	20	16	4096	694
	30	16	4096	886
	50	16	4096	1272
	100	16	4096	2235
	120	16	4096	2620

For any number of nodes, Floyd's algorithm consistently requires more iterations than Dijkstra's algorithm, making it the more time-consuming approach. However, we found some limitations when implementing Dijkstra's algorithm in SAS. In contrast, Floyd's and Floyd-like algorithms require three simple DO loops and can be easily implemented using SAS/IML. Consequently, we decided not to write code for generating the implicit edit bounds using Dijkstra's algorithm.