PROFILING AND REBUNDLING MEDICAL CARE CLAIMS USING THE FORMAT PROCEDURE AND MACROS

Michael Davis, Blue Cross & Blue Shield of Connecticut, Inc.

Abstract

Medical care providers commonly resort to billing separately for procedures which should be billed as a single CPT code. Writing a program to profile or rebundle medical claims is complicated by problems such as the:

- need to evaluate a random number of claims simultaneously
- permutations of logic used to recombine claims
- characteristics of profiles or bundles that are not composed of mutually exclusive CPT codes

An elegant solution to these difficulties is to use some of the subtle features of the FORMAT Procedure and Macros. The following describes how to write a program which may be used under Releases 5.18, 6.04, or Version 6.06.

Introduction

Medical procedures, such as physician and laboratory services, are often categorized and paid using a coding system known as CPT (Current Procedural Terminology), which was developed and is maintained by the American Medical Association. CPT codes are 5 characters in length and may include one or two modifiers, which are 2 characters long.

Some procedures are performed in organ-related groups, especially laboratory tests performed on automated analyzers. These groups of procedures are known as "panels" or "profiles". Accordingly, healthcare payors often structure their reimbursement contracts to pay for the CPT code that represents the profile rather than for the components of the profiles.

Since the profile payment is usually less than the sum of the payments for individual tests, there is an economic incentive for providers to bill for individual tests. Also, since billing procedures vary among payors, a provider may decide to bill for individual tests and leave the payor with the responsibility for assigning the tests to a profile.

In January 1988, Blue Cross & Blue Shield of Connecticut, Inc. began to contract with participating clinical laboratories to reimburse them using the profiles contained in the Pathology and Laboratory section of CPT. Largely because of the complexity of programming the profile payments, it was not possible to program the claims processing system to bundle individual procedures into profiles in 1988.

Programming Complications

The construction of laboratory profiles presents several programming complications. First, the CPT codes composing laboratory profiles are not mutually exclusive. For example, CPT code 83718, HDL cholesterol, is used to construct profile 80061, lipid profile and profile 80062, cardiac evaluation panel. Thus, a profiling program must keep track of whether a procedure has been used to construct a profile or is still available.

This leads to the second complication. Since a CPT code assigned to one profile may prevent some other profile from being constructed, then the order in which profiles may be evaluated is critical. Improper sequencing of the tests for each profile could lead to the creation of "phantom profiles," where a profile would exist on the schedule of payment allowances but no claims payments would be made for it.

The second complication leads to a third, which is the need to construct the program so the sequence of profile tests can be reordered quickly and easily. If the program

1 This raises the question, why may a code not be reused (similar to the issue of replacement when considering probability)? Perhaps some reimbursement programs would permit reuse. The major reason why reuse is not permitted here is that it might lead to paying for a CPT code twice, potentially increasing total reimbursement.

2 This program permits CPT codes to be repeated on the same claim. This is because sometimes, a CPT code may be used to represent different procedures.
code is not modular, it will be difficult to modify the program to accommodate problems discovered through testing.

Fourth, the number of procedures on a claim varies. During testing, it was discovered that the number of CPT codes on a claim was as many as 59. There is no maximum limit to the number of procedures that can be examined.

Fifth and last, because of the recursive nature of the testing algorithm, and the millions of claims to be examined, techniques that reduced the frequency of reading and writing to disk were needed to minimize runtime.

Subtle Features of PROC FORMAT

Using FORMAT procedure as a table-lookup technique provides several advantages valuable for programming this type of project. First, if a value is not accounted for in the input ranges, it is not changed. Additional programming is not required to handle codes extraneous to the profile or to test whether the codes on the claim have been profiled.

Second, the FORMAT procedure incorporates shortcuts for defining the codes to be formatted. For example, a long series of codes may be defined as a range. The ability to define the ranges with key words (LOW, HIGH, and OTHER) permits even more compact program code.

Third, since the program code is modular and the required formats are similar, much of the format coding may be reused. This reduces coding time and program size considerably.

Fourth, the SAS compiler flags duplicate arguments within a single VALUE statement. This adds protection from a common program coding error.

Last, the FORMAT procedure compiles the look-up table into an efficient decision-tree structure that may be retained in a permanent library. This greatly reduces execution time on a personal computer, which is a major advantage when testing.

Simplifying the Combination Logic

The conceptual breakthrough, which made the coding and testing of this program vastly easier, was defining the profiles as either a counting test or an "all" test.

The Count Test:

\[ \text{True if } (A,B,C, ... ,Z) \geq \text{Cutoff} \]

where \((A,B,C, ... ,Z)\) represents how many codes from a list of CPT codes are present on the claim being evaluated and Cutoff represents the number of procedures required to compose the profile being evaluated.

The All Test:

\[ \text{True if } (A \text{ and } B \text{ and } C \text{ ... and } Z) \]

\((A \text{ and } B \text{ and } C \text{ ... and } Z)\) is a list of CPT codes, all of which must be present for the test to be evaluated as true.

All of the required profiles could be constructed by chaining and sequencing the above two tests. An "or" test was not necessary since "or" conditions could be expressed through the Count Test as shown below:

\[ \text{True if } (A,B,C, ... ,Z) \geq 1 \]

Reducing the Volume of Code Through Macros

Defining the program as the repeated application of two logical tests (Count and All) leads to the recognition that the SAS macro facility could significantly reduce the size of the program code, increase the modularity of the program and facilitate debugging.

\[ \text{4 It is worth noting that while there was no need to evaluate negative conditions (that a CPT code was not present) for the laboratory profiles, the addition of a "Not Count Test" and a "Not All" would accommodate that requirement.} \]

\[ \text{5 An additional logical test combining count with a test for a single CPT code was also used in this program.} \]
%CNT and %ALL are the macros which perform these two tests.

The macro for the Count Test (%CNT) was coded as:

```
macro CNT(CUTOFF,CTEST,PCODE,SRC,WO);
co/o/oeval(&CUTOFF);
SUM= 0;
do I= 1 to COUNT;
   if %upcase(&SRC) = "WORK"
      then CNTTST{I} = put(WORK{I},S&CTEST..);
   else CNTTST{I} = put(PCODES{I},S&CTEST..);
   SUM= SUM+INPUT(CNTTST{I},1.0);
end;
if SUM ge CO then do;
do I= 1 to COUNT;
   if %upcase(&WO) = "OUT"
      then PCODES{I} = put(WORK{I},$&PCODE..);
   else if %upcase(&WO) = "OUT"
      then PCODES{I} = put(PCODES{I},$&PCODE..);
   else WORK{I} = put(PCODES{I},S&PCODE..);
end;
end;
mend CNT;
```

If the profile can be evaluated by a single iteration of %CNT, then &SRC is set to "NULL" and &WO is set to "OUT". The format named by &CTEST puts a value of 1 or 0 in as many elements of the CNTTST array as there are CPT codes. If the sum of CNTTST is greater than &CUTOFF, then the CPT codes in the INPUT array are translated by the format named by &PCODE and placed in the PCODES array to be written as profiled observations.

If %CNT is chained to future executions of %CNT or %ALL, then &WO is set to "WORK". This directs %CNT to translate the CPT codes in the INPUT array using the format named in &PCODE, and then to place them in the WORK array. Subsequent %CNT and %ALL iterations are set to test against the CPT codes in the WORK array rather than the INPUT array since the value of &SRC is set to "WORK". The final macro iteration is directed to write out the profiled observations by setting the value of &WO to "OUT".

The ALL Test (%ALL) is similar to the Count Test (%CNT) but is more involved. It is coded as follows:

```
macro ALL(CUTOFF,CTEST,PCODE,SRC,WO);
CO= %eval(&CUTOFF);
do I= 1 to CO;
   ALLTST{I} = 0;
end;
do I= 1 to COUNT;
   if %upcase(&SRC) = "WORK"
      then J= input(put(WORK{I},S&CTEST..),1.0);
   else J= input(put(PCODES{I},S&CTEST..),1.0);
   ALLTST{J} = 1;
end;
SUM= 0;
do I= 1 to CO;
   SUM= SUM+ALLTST{I};
end;
if SUM ge CO then do;
do I= 1 to COUNT;
   if %upcase(&WO) = "OUT"
      then PCODES{I} = put(WORK{I},S&PCODE..);
   else if %upcase(&WO) = "OUT"
      then PCODES{I} = put(PCODES{I},S&PCODE..);
   else WORK{I} = put(PCODES{I},S&PCODE..);
end;
end;
mend ALL;
```

The workings of the %ALL macro are better understood by looking at two examples. For the first example, assume that the logical test to be performed is:

```
82756 if 84479 and 84436
```

and the CPT codes on the claim are 82040, 84479, and 84436. The format named by &CTEST causes the 84479 to set ALLTST{1} to 1 and causes the 84436 to set ALLTST{2} to 1. The format does not list 82040 so the format arbitrarily causes all other codes to set ALLTST{3} to 1.

Since &CUTOFF is set to 2, when ALLTST{1} and ALLTST{2} are summed, they equal &CUTOFF and the format named by &PCODE translates 84479 and 84436 as 82756. CPT code 82040 remains unchanged.

One subtle feature of this approach is that it handles combined AND and OR tests with a minimum of program code.
For example, assume that the logical test to be performed is:

99999 if 84443 and (84479 or 82756)

and the CPT codes on the claim are 84443 and 84479. 99999 is an arbitrary code used when a test will be chained to one or more subsequent tests to make it easier to spot coding errors. The format named by &CTEST will set ALLTST(2) to 1 when either 84479 or 82756 appears on a claim. The test will be evaluated as true and 99999 will be written out as an intermediate profiled code.

The All Test (%ALL) also can be used to test for a single code, which can then be translated into another code. Also, %ALL and %COUNT can test for one set of CPT codes yet translate another set of CPT codes which support some very complicated profile definitions.

The previous examples show the flexibility of this macro/format character translation technique. The sequence of the tests can be reordered quickly with a line editor, and the macros allow infinite flexibility in constructing profiles.

Testing the Logic on a Personal Computer

The testing of the logic of the program was accomplished on a personal computer running Release 6.04 under PC-DOS. The personal computer was selected for composing and testing the program to take advantage of more flexible program editors and print formatters available on that platform as well as the better version of Display Manager. The two issues which needed to be satisfied before testing with actual data was that the assignment of codes was as specified and that there were no phantom codes.

To accomplish this, a test claim dataset was constructed so that at least one claim exercised each branch of the profiling logic. To speed the detection of problems, one of the ID variables in the output dataset was used to show the CPT code which the profiled code should have resolved to. By comparing the ID field with the profiled code field, problems were quickly identified.

While this approach was a good start, a shortcoming of this testing method eventually emerged. While this form of testing correctly identified false negative situations, it offered no protection against false positive situations. A false positive situation is where the CPT codes on a claim are incorrectly translated into a profile code instead of being ignored.

Differences in SAS Across Platforms

Memory shortages were encountered while testing the code on a personal computer under Release 6.04. However, it was possible to complete the initial tests by dropping Display Manager and running SAS in batch mode. This problem was not given much attention since the production version of the program would be run under Version 5.18 on an OS mainframe. Since the OS mainframe had many times more memory than was available on the personal computer, it was felt that there would be no problem in porting the application.

Ironically, when the program was executed on OS mainframe, it could not be compiled. SAS Institute's Technical Support Department indicated that Version 5.18 allocates an arbitrarily assigned, finite region for compilation. Since the macro compiler was generating several thousand lines of program code within a single data step, the available region was exceeded.

Fortunately, the modular design of the program offered a quick solution. The single data step was split into four and the results from the preceding steps were fed into the next step. Some efficiency was lost by the need to write the intermediate results to disk. Release 6.06 for OS mainframes lifts the restriction on compilation region size.

Only one syntax difference emerged when the program was ported to Release 5.18. Several RETAIN statements were used to preserve the values in the arrays while executing a data step multiple times for the same claim. They had been originally coded with the array name under Release 6.04 and the program had compiled successfully. When the program was ported to Version 5.18, it was necessary to recode the RETAIN statements to use the variable ranges instead of the array names.

Tips for Working with Clients

One lesson that emerged from this project is that clients without programming experience may not be able to develop complete program specifications. For example, until the test runs were shared, the impact of the order in which the profile tests were executed had not been fully contemplated. Another unanticipated consideration was how to handle claims which contain both profile codes and CPT codes.
From the time the results of the test program were shared with the client until the program was fully approved, the number of macro tests was doubled to accommodate the needs which had not been anticipated.

However, if one brings a positive attitude towards working with clients to a project, there is much to be gained. For example, the clients on this project found many abnormalities, some of which often turned out to be coding errors.

When the project was first conceived, no information regarding the original code was to be placed on the output SAS dataset since a profile code summarized an indeterminate number of CPT codes. However, one of the clients suggested that the program output an observation for each CPT code on the claim. Thanks to this suggestion, an effective audit trail was created.

Testing With Claim Data

Testing the program with actual claim data resulted in the detection of false positive results, SAS coding discrepancies, improper test sequence, and confusion about how the profiles were to be tested, as noted earlier. However, two additional types of situations were identified at this stage.

It had been assumed that translating a series of CPT codes into a profile always resulted in a smaller payment to the laboratory. However, several cases were discovered where the payment amount was greater. The overall reimbursement impact was insignificant.

Also, it was discovered that in several cases, both the profile code and some of the component CPT codes appeared on the same claim.

For example, a laboratory might bill CPT code 80050, general health screen profile with a modifier code of 15, indicating one additional test. On the same claim, the CPT code 83718, HDL cholesterol, also would appear. Assuming that 83718 had not been already included in CPT code 80050.15, the two codes should have been combined as 80050.16.

The problem was overcome by testing for the above and similar combinations. In order not to prematurely eliminate any possible combinations, the testing was done recursively. In the above example, only the 80050.15 was translated during the first pass to CPT code 80050.16. On the final pass, 83718 was translated to 80050.16 as well.

Building Claims Tracing Facilities (Audit Trail)

As mentioned earlier, by including the inputted CPT code as well as the translated CPT code as variables on the outputted observation, a rudimentary audit trail is provided. For the laboratory profile application, in which no CPT code needed to be translated more than twice, this type of audit trail was sufficient.

However, it is conceivable that a more extensive audit trail could be required, especially if multiple translations of particular CPT codes are required. This could be accomplished by a log which outputted a record with the input CPT code, the translated CPT code, and identifying information. One record would appear for every test where translation occurred.

Another approach would be to add variables to the output SAS dataset which would be populated with the various generations of input CPT codes which were subsequently translated. While the SAS code for this approach is more involved, such an audit trail is easier to follow.

Conclusion

The macro/format approach proved to be a good way to profile laboratory claims and would probably work equally well for any application requiring the rebundling of CPT codes. In addition to the macros and the formats, program code was written to read in the claims data, write the profiled CPT codes to disk, and reprice the claims which had been translated to new profile codes.

For the 30-months beginning in January 1988, more than $1 million of potential savings were identified by this computer program for participating laboratory claims.

Epilogue

Since this paper was presented at SUGI 16, the application was rewritten to take advantage of the feature of Release 6.06 which permits compilation of data steps of virtually unlimited size. It was expected that with the four main data steps combined into one, execution time would be significantly reduced.

Surprisingly, when the Release 6.06 version ran, it took nearly twice the execution time and nearly ten times as
much region as the Release 5.18 version of the program! When the Technical Support Department at SAS Institute was contacted, they noted that many program elements used this program executed more slowly under Release 6.06 than under Release 5.18. These include:

- Informats/Formats
- Functions
- Character to numeric conversions
- PROC FORMAT style formats
- Division by 0
- Macro facility

SAS Institute indicated that Release 6.07 of the SAS System should offer significantly improved performance of programs using these features.

REFERENCES


ACKNOWLEDGEMENT

The author expresses his gratitude to Vincent Maffei, Louise Shanaman, Linda Cavanagh, Karen Clarke, and Renee Randall, all of Blue Cross & Blue Shield of Connecticut, Inc., who collaborated with him on this project. He would also like to thank the Technical Support Department at the SAS Institute.

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