Paper 242-27

The Heuristic Program

John R. Gerlach, NDC Health;Yardley, PA

Abstract

Most data analysis problems require more time and effort than usually allotted. Consequently, the highly overrated “quick and dirty” solution often fails when implemented under real conditions. Because of typical demanding circumstances, it is a good idea to develop a method for solving problems expeditiously, albeit comprehensively. This paper discusses heuristic programming as a method to solve problems effectively, especially those so-called “quick questions.”

Introduction

Every data analyst has experienced the notorious “quick question” and the undocumented “quick and dirty” solution. You know the scenario: It’s an ‘easy’ problem (which nobody truly understands); yet, somehow, the solution should not take long to write. Then, of course, when the solution fails under real conditions, there’s hell to pay. Certainly, there’s a better approach, even with unreasonable circumstances.

Consider taking a heuristic approach to problem solving, rather than suffering the shotgun approach. The heuristic method is a problem solving technique that promotes study and investigation of the problem such that the most appropriate solution is selected at successive stages of development.

Writing a heuristic program allows the analyst / programmer to study the problem in depth, as well as to solve the problem more rigorously. In fact, the process resembles a designed experiment, rather than a hasty crap shoot. Consequently, the proposed solution evolves into something more robust and efficient as compared to an ill-fated solution.

The Method

The scenario for writing a heuristic program is:

- Generate contrived data such that you can anticipate the results.
- Produce listings throughout the program to validate the process.
- Inspect the results (including the LOG).
- Make modifications in an attempt to break or to improve your solution.

The following situations illustrate this problem-solving approach.

1. Transposing Data

Transposing data is, perhaps, a classical example of underestimating a problem and, thereby, trivializing its solution. Consider using a heuristic approach to study this common task by first creating contrived data that will anticipate the results, as follows.

```
data orig;
   do presc = 1 to 5;
      do product = 1 to 3;
         if ranuni(0) lt 0.75
            then do;
               ntrx = int(ranuni(0)*10);
             output;
             end;
         end;
      end;
run;
```

```
Obs    presc    product    ntrx
  1      1         1         7
  2      1         2         5
  3      1         3         2
  4      2         1         5    ←
  5      2         3         8    ←
  6      3         1         6
  7      3         2         5
  8      3         3         7
  9      4         1         5    ←
 10      4         2         9    ←
 11      5         1         4
 12      5         2         7
 13      5         3         3
```

Notice that plans #2 and #4 do not have all possible products. Of course, the transpose process should consider that situation, as well. Now, consider the first solution and its output.

```
proc transpose data=orig prefix=ntrx
   out=method1(drop=_name_);
   var ntrx;
   by presc;
run;
```

```
presc    ntrx1    ntrx2    ntrx3
  1        7        5        2
  2        5        8        .    ←
  3        6        5        7
  4        5        9        .    ←
  5        4        7        3
```

The first solution fails to process correctly those plans not having all possible drugs represented. Upon doing a bit of research (e.g., reading the manual), one learns that the ID statement corrects this problem.

```
proc transpose data=orig prefix=ntrx
   out=method1(drop=_name_);
   id product;
   var ntrx;
   by presc;
run;
```

```
presc    ntrx1    ntrx2    ntrx3
  1        7        5        2
  2        5        .        8    ←
  3        6        5        7
  4        5        9        .    ←
  5        4        7        3
```

Still, is this solution sufficient? Perhaps the missing values are inappropriate for the intended analysis. Thus, it may be better to use a Data step to transpose the data and to re-code missing values to zero, as follows.

```
data new2;
   array ntrx_{*} ntrx1-ntrx3;
   retain ntrx1-ntrx3;
   set orig;
      by presc;
   if first.presc
      then do i = 1 to dim(ntrx_);
         ntrx_{i}=0;
         end;
   ntrx_{product} = ntrx;
   if last.presc then output;
   keep plan ntrx1-ntrx3;
run;
```

The listing below shows the effect of this alternative solution. Notice that the aforementioned Data step must know the number of plans *a priori*; whereas, the Transpose procedure did not.

```
presc    ntrx1    ntrx2    ntrx3
  1        7        5        2
  2        5        0        8
  3        6        5        7
  4        5        9        0
  5        4        7        3
```

2. Selecting the top 25 whatever

The next problem concerns a large population of physicians each assigned to one of several geographical areas, called a territory. Imagine that you want to generate a subset data set, those physicians who prescribe the most often, for each territory. Assume that you do not have access to any real data; rather, you are asked to solve the problem. Thus, you generate contrived data emulating the problem, as follows.

```
data orig;
   do territory = 1 to 5;
      do presc = 1 to 50;
         prescriber = put(territory,1.)
               || put(presc,z3.);
         nrx = int(ranuni(0)*10)*presc;
         output;
         end;
      end;
   drop presc;
run;
```

The listing below represents only one territory, but is sufficient for our discussion of the problem. Notice that some prescribers have zero Rx scripts. Clearly, in this case, these physicians not become part of the subset population. Also, notice that several physicians have the same number of scripts.

```
Presc  # Rx    Presc  # Rx    Presc  # Rx
-----------    -----------    -----------
 1001     5     1014    84     1027    81
 1002    18     1015   120     1028    56
 1003    24     1016    80     1029    29
 1004    20     1017    99     1030   180
 1005    45     1018   144     1031   217
 1006    54     1019    38     1032   192
 1007    49     1020    80     1033    99
 1008     8     1021   168     1034    34
 1009    54     1022    44     1035   140
 1010    30     1023     0     1036   144
 1011    99     1024   216     1037    74
 1012   108     1025   225     1038   228
 1013     0     1026   130
```

A good idea: Simply sort the data set by territory and by the number of scripts (in descending order), then use by-group processing in a subsequent Data step, taking the first 25 observations of each group.

```
proc sort data=orig out=srtd;
   by territory descending nrx;
run;
```

The BY statement of the SORT procedure uses the DESCENDING option to order the data set as needed. The Data step uses FIRST. logic to initialize a counter that determines whether the physician is selected. The counter is incremented using a SUM statement. The Data step is shown below.

```
data top25;
   set srtd;
      by territory descending nrx;
   if first.territory
      then n=1;
   if n le 25 then output;
   n+1;
run;
```

Notice that this solution does not care how many physicians are in a given by-group. But, what about those physicians having the same number of scripts that denotes the cutoff point, in this case 99 scripts? Does it matter who gets selected out of that group? Perhaps, a random selection of the tie group is required, which poses a much bigger problem.

```
Presc   # Rx    Presc   # Rx    Presc    # Rx
-----------     -----------     ------------
1040     320    1030     180    1046      138
1043     301    1045     180    1044      132
1041     246    1021     168    1026      130
1038     228    1050     150    1015      120
1025     225    1018     144    1039      117
1031     217    1036     144    1012      108
1024     216    1048     144    1011       99   ←
1042     210    1047     141
1032     192    1035     140
```

When writing heuristic programs to solve data analysis problems, it is important to corroborate your results. In this example, it is obvious that the target data set should contain no more than 25 physicians for each territory. The FREQ procedure can be used to produce a distribution of the variable denoting territory, as follows.

```
proc freq;
   tables territory;
run;
                                      Cumulative
territory    Frequency    Percent      Percent
------------------------------------------------
        1           25      20.00        20.00
        2           25      20.00        40.00
        3           25      20.00        60.00
        4           25      20.00        80.00
        5           25      20.00       100.00
```

3. Preparing Data for Logistical Regression

Assume you are told about, not given a copy of, a SAS data set, containing only five observations, that looks like the following.

```
 m1     m2     m3     m4     m5     m6    m7
 70     60     50     40     30     20    10
140    120    100     80     60     40    20
210    180    150    120     90     60    30
280    240    200    160    120     80    40
350    300    250    200    150    100    50
```

Even worse, from the ‘given’ data set, you are asked to produce the following data set needed to perform logistical regression analysis.

```
mth   count   total   T1    T2    T3    T4    T5

 1     70      70      1     0     0     0     0
 2     60      70      1     0     0     0     0
 3     50      70      1     0     0     0     0
 4     40      70      1     0     0     0     0
 5     30      70      1     0     0     0     0
 6     20      70      1     0     0     0     0
 7     10      70      1     0     0     0     0
 1    140     140      0     1     0     0     0
 2    120     140      0     1     0     0     0
 3    100     140      0     1     0     0     0
 4     80     140      0     1     0     0     0
 5     60     140      0     1     0     0     0
 6     40     140      0     1     0     0     0
 7     20     140      0     1     0     0     0
```

Notice that the variable M1 of the input data set becomes the variable TOTAL in the target data set. Yet, all the variables, M1-M7, are transposed to populate the variable COUNT. The variables M1-M7 represent seven months of patient counts which maps to the variable MTH in the target data set explicitly denotes the month. Finally, the variables T1-T7 are dichotomous variables denoting the *i*th observation of the input data set. Thus, for example, the variable T1 appropriately contains the value 1 and the respective variables T2-T7 are assigned the value 0 for the first seven observations.

The following Data step creates the input data set that emulates the problem. It’s no coincidence that the variables M1-M7 for a given observation decrease in value. In fact, the variable M1 denotes the total number of patients; whereas, the variables M2 –M7 represent the attrition of those patients over time.

```
data orig;
   array mths{*} m7 m6 m5 m4 m3 m2 m1;
   do i = 1 to 5;
      do j = 1 to dim(mths);
         mths{j} = i*(j*10);
         end;
      output;
      end;
   drop i;
run;
```

The solution employs a single Data step that processes the contrived input data to produce the desired target data set. Rather than expound on the code, it's important to realize the method that expedited a good solution to a difficult problem.

```
data revised;
   array types{*} type1-type5;
   array mths{*} m1-m7;
   set orig;
   do i = 1 to dim(types);
      types{i}=0;
      end;
   total = m1;
   do mth = 1 to dim(mths);
      count = mths{mth};
      types{_n_} = 1;
      output;
      end;
   keep mth count total type1-type5;
run;
```

4. Splitting a Large SAS Data Set

Suppose you have a very large data set that you wish to partition into *n* smaller data sets having comparable cardinality. Obviously, a Data step having appropriate logic can partition any data set, accordingly. Idea: Automate that process.

```
%macro split(ndsn=2);
   data %do i = 1 %to &ndsn.; dsn&i. %end; ;
      retain x;
      set orig nobs=nobs;
      if _n_ eq 1
         then do;
            if mod(nobs,&ndsn.) eq 0
               then x=int(nobs/&ndsn.);
               else x=int(nobs/&ndsn.)+1;
            end;
      if _n_ le x then output dsn1;
      %do i = 2 %to &ndsn.;
         else if _n_ le (&i.*x)
            then output dsn&i.;
         %end;
   run;
%mend split;
```

Again, rather than expound on the code, it's more important to understand a method of problem solving and to realize, for instance, that this problem offers a nice exercise in basic number theory, as well as Boolean logic.

As always, test the code. In this case, it's a good idea to use the MPRINT option to observe how the macro resolves. For the test, the following Data step generates a simple data set containing eight-two observations.

```
options mprint;
```

```
data orig;
   do i = 1 to 82; output; end;
run;
```

The first invocation splits the 'large' data set into two data sets each containing forty-one observations, as expected. The SAS log shows the results.

```
%split(ndsn=2)  ;

WORK.DSN1 has 41 observations and 2 variables.
WORK.DSN2 has 41 observations and 2 variables.
```

The second invocation splits the same data set into four data sets. Observe that the fourth data set contains only nineteen observations, which is reasonable.

```
%split(ndsn=4)   ;

WORK.DSN1 has 21 observations and 2 variables.
WORK.DSN2 has 21 observations and 2 variables.
WORK.DSN3 has 21 observations and 2 variables.
WORK.DSN4 has 19 observations and 2 variables.
```

The final invocation presents a caveat in the solution worthy of some discussion. Assuming that it is reasonable to split a data set having 82 observations into 43 smaller observations, the %split macro creates two extraneous data sets containing zero observations.

```
%split(ndsn=43)  ;

WORK.DSN1 has 2 observations and 2 variables.
WORK.DSN2 has 2 observations and 2 variables.
WORK.DSN3 has 2 observations and 2 variables.
        :      :      :      :      :
WORK.DSN40 has 2 observations and 2 variables.
WORK.DSN41 has 2 observations and 2 variables.
WORK.DSN42 has 0 observations and 2 variables.
WORK.DSN43 has 0 observations and 2 variables.
```

Conclusion

Writing a heuristic program allows the analyst / programmer to study a problem thoroughly, as well as to solve the problem more rigorously. In fact, the process becomes more like an experiment, rather than a hurried response. That is, the solution evolves into something more elegant and robust. Most importantly, perhaps, this technique promotes learning and development.

Author Information

John R. Gerlach; NDC Health; Yardley, PA.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.