Paper 5001-2020

Make Your DO Loop More Efficient

G Liu

ABSTRACT

In certain cases, a DO loop does not need to run all of its iterations in order to obtain the correct result. This paper will examine three examples to illustrate when we can or cannot make the DO loop more efficient with no loss of accuracy, through the use of the LEAVE statement.

INTRODUCTION

Time is money. An efficient SAS® code not only helps us obtain the result faster, increases our bottom line, and gives us an edge over our competitors, it also utilizes less computing resources thus saves the environment. By using the LEAVE statement in certain DO loops, the same accurate result can be achieved efficiently.

In cases where an indicator flag (or indicator variable) with a value of either 1 or 0 is determined based on multiple variables through the use of a DO loop, the DO loop may not need to run all its iterations to obtain the correct 1 or 0 value. As soon as the indicator flag has been assigned the desired value at certain iteration, the remaining iterations can be skipped by properly adding the LEAVE statement to the DO loop, thus eliminating unnecessary processing and shortening runtime.

Consider the following equivalent idea in the more familiar form of an OR statement:

```
if brain_cancer=1 or lung_cancer=1 or skin_cancer=1 then cancer=1;
```

Or in the form of IF-ELSE IF statements:

```
if brain_cancer=1 then cancer=1;
else if lung_cancer=1 then cancer=1;
else if skin_cancer=1 then cancer=1;
```

As long as the *brain_cancer* variable has a value of 1, the indicator flag *cancer* will be assigned the value of 1 as well, regardless of the values in the variables *lung_cancer* and *skin_cancer*. Through the use of the OR or the ELSE IF statements, SAS does not need to process the two latter parts of the logic in order to obtain the correct value of 1 for the *cancer* variable.

When the "parent" variables (e.g. *brain_cancer, lung_cancer* etc.) are placed in an ARRAY, a DO loop processing is necessary to loop through all of them. The SAS code in the form of an ARRAY and DO loop:

```
array cancer_type(*) brain_cancer lung_cancer skin_cancer;
do i=1 to dim(cancer_type);
    if cancer_type(i)=1 then cancer=1;
end;
```

However, notice that this DO loop will always iterate three times, once for each of the cancer types, regardless of the values in the "parent" variables. This is essentially three separate IF statements rather than the more efficient IF-ELSE IF version. By adding the LEAVE statement inside the DO loop, unnecessary iterations can be skipped. The following

three case studies detail whether the LEAVE statement can be used to improve efficiency while maintaining the accuracy of the result.

CASE STUDY 1

A hospital administrator is interested in finding out the number of cancer patients the hospital has treated. The cancer diagnosis can be coded in one or more of the 25 diagnosis code variables (*dx1 - dx25*) in the data. In order to identify the cancer patients, the SAS code needs to loop through all 25 variables, and as long as any of them has a cancer diagnosis, that patient is to be counted. (Cancer diagnosis codes start with 'C00' to 'D49'.)

The less efficient version of the code:

```
array dx(*) dx1-dx25;
do i=1 to dim(dx);
    if 'C00' le substr(dx(i),1,3) le 'D49' then cancer=1;
end;
```

Table 1 shows the resulting dataset, with variable *i* showing the value of 26 for all three observations. This means that each observation has been iterated 25 times through the DO loop, and when variable *i* is incremented to the value 26, the DO loop logic no longer applies and the DO loop stops after 25 iterations. The shaded values are the cancer diagnosis codes.

dx1	dx2	dx3	...	i	cancer
A35.1	C30.24		...	26	1
Y02.353E	Z02.119		...	26	.
D21.12	D22.10	E32.19	...	26	1

Table 1. Dataset after the less efficient coding is run

The more efficient version of the code:

```
array dx(*) dx1-dx25;
do i=1 to dim(dx);
    if cancer=1 then leave;
    if 'C00' le substr(dx(i),1,3) le 'D49' then cancer=1;
end;
```

By adding the LEAVE statement right after the DO statement to check whether the *cancer* variable already has a value of 1, as soon as *cancer* is set to 1, any subsequent iterations are skipped, and the processing leaves the DO loop. Whether or not there are more cancer diagnoses in the later diagnosis codes are irrelevant, and the *cancer* value of 1 can no longer be changed.

Table 2 shows the resulting dataset after running the more efficient version of the code with the LEAVE statement added to the DO loop.

dx1	dx2	dx3	...	i	cancer
A35.1	C30.24		...	3	1
Y02.353E	Z02.119		...	26	.
D21.12	D22.10	E32.19	...	2	1

Table 2. Dataset after the more efficient coding is run

Note that the variable *i* has much smaller values for the two observations with cancer diagnosis. On the first observation, since *dx2* is a cancer diagnosis, *cancer* is set to 1 after the second iteration. At the beginning of the third iteration (*i=3*), the LEAVE statement applies and the remaining iterations are not executed. On the other hand, the second observation has no cancer diagnosis, hence all 25 iterations are performed and the LEAVE statement is never applied.

ADDITIONAL EFFICIENCY GAIN WITH DOMAIN KNOWLEDGE

Healthcare data experts understand that as soon as a blank value is encountered on a diagnosis code (e.g. *dx3*), the remaining diagnosis codes (*dx4 – dx25*) all have blank values as well. All those iterations checking on blank values can be skipped as well, further improving the efficiency of the code.

The best code accounting for blank values:

```
array dx(*) dx1-dx25;
do i=1 to dim(dx);
    if cancer=1 or dx(i)='' then leave;
    if 'C00' le substr(dx(i),1,3) le 'D49' then cancer=1;
end;
```

CASE STUDY 2

Similar to Case Study 1, if the hospital administrator is interested in identifying digestive and bone cancer patients separately, the LEAVE statement will need to check both indicator flags before leaving the DO loop.

The efficient code:

```
array dx(*) dx1-dx25;
do i=1 to dim(dx);
    if digestive=1 and bone=1 or dx(i)='' then leave;
    if 'C15' le substr(dx(i),1,3) le 'C26' then digestive=1;
    if 'C40' le substr(dx(i),1,3) le 'C41' then bone=1;
end;
```

Note that the reason the LEAVE statement ought to check for both flags is because if only one of the flags has a value of 1, the DO loop needs to continue checking the remaining diagnosis codes to see if the other flag needs to be set to 1 as well. However, as soon as both flags are set to 1, there is no longer any need to check the remaining diagnosis codes because the values of both flags will no longer change.

CASE STUDY 3

The hospital administrator is interested in finding patients with knee replacement surgery but without a revision surgery at a later date. The surgery data consist of 40 procedure code variables (*px1 – px40*) sorted in ascending order of procedure date, i.e. procedure in *px2* is performed after *px1, px3* performed after *px2*, and so on. Assume that we use the value 'OSP' and 'OSW' to identify knee replacement and revision respectively.

The SAS code to find these patients:

```
array px(*) px1-px40;
do i=1 to dim(px);
    if surgery=1 then leave;
    if      substr(px(i),1,3)='OSP' then surgery=1;
    else if substr(px(i),1,3)='OSW' then surgery=0;
end;
```

Notice that in this case the LEAVE statement cannot be added to the DO loop. Even after the *surgery* flag has been set to 1 when a knee replacement surgery is found, SAS still needs to check all the remaining procedure codes to make sure that a knee revision surgery is not found after the knee replacement.

CONCLUSION

The LEAVE statement is a useful tool for controlling whether certain iterations of the DO loop can be skipped. In cases where the value of an indicator flag is set, and any subsequent iterations can no longer change this value, then completing all iterations of the DO loop is unnecessary. By adding the LEAVE statement and checking for the proper condition to leave the DO loop, fewer iterations are needed to obtain the same result, thus achieving efficiency and reducing runtime.

REFERENCES

Nistor, A., Chang, P., Radoi, C., Lu, S. 2015. "CARAMEL: Detecting and Fixing Performance Problems That Have Non-Intrusive Fixes." *Proceedings of the 37th International Conference on Software Engineering – Volume 1*, 902–912.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

G Liu
omnibus.g.liu@gmail.com