Paper 10641-2016

The Path Length: Parent Child De-lineage with PROC TREE and ODS

Can Tongur, Statistics Sweden

ABSTRACT

The SAS® procedure PROC TREE sketches parent-child lineage—also known as trees—from hierarchical data. Hierarchical relationships can be difficult to flatten out into a data set, but with PROC TREE, its accompanying ODS table TREELISTING, and some creative yet simple handcrafting, a de-lineage of parent-children variables can be derived. Because the focus of PROC TREE is to provide the tree structure in graphical form, it does not explicitly output the depth of the tree, although the depth is visualized in the accompanying graph. Perhaps unknown to most, the path length variable, or simply the height of the tree, can be extracted from PROC TREE merely by capturing it from the ODS output, as demonstrated in this paper.

INTRODUCTION

A frequently used data structure in relational databases is hierarchical relationships between variables: one variable is the parent of another variable whence is the child of the parent.

Hierarchical data structures may be found in many contexts. For instance, a Consumer Price Index (CPI) is an aggregated index value consisting of several sub-indexes, all being mutually exclusive at each level, i.e. with unique kinships and straightly descending.

Alternatively, patients in care at hospitals can be described as being in a hierarchical structure starting with the patient that belongs to one or more physiologists that in turn are enrolled at different departments in one or more hospitals. In such a case, kinships are non-unique and nested and cycles would occur in the graph or tree structure. Then, kinships would have two or more directions, as known from graph theory (see e.g. Koski & Noble, 2009).

Another way of considering the case of unique descending kinships is through graph notation and, more familiarly, as a tree structure. The root of the tree, a root node, is a parent variable spawning its descendants – children variables which themselves may be parents – as branches and leaves of the tree. A graph having unique kinships can be referred to as a-cyclical: all paths between the root and the leaves, i.e. the top parent node and the descending children nodes, are unique and ending – no nodes are pointed back upwards in the hierarchy such that a loop occurs when there are two parents to any child node. A pedagogical introduction to trees in SAS can be found in Nizol (2010) in which pruning and tree traversal are explained.

Hierarchical data most often need to be “flattened out” in order to be processed, which can be done by an algorithmic approach as described in Nizol (2010). However, this requires a more profound comprehension of programming, which many of us may be lacking.
Fortunately, the SAS® procedure PROC TREE comes to our rescue.

EXAMPLE: A TRIVIAL TREE DATASET

The dataset HIERARCHY contains a simple hierarchy embodied in two columns: Parent and Child. The top node is 1 and this has no parent, hence the leading null value on the first row in the Parent column. Below that node, the tree structure entangles from the tree root/top node 1 to its branches and leaves, i.e. to all descendants in the parent-child hierarchy.

In Figure 1 the parent-child concept is sketched for the example:

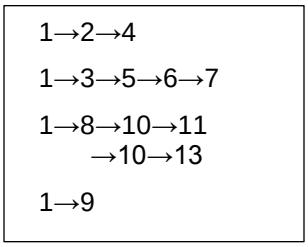


Figure 1. An exemplified parent-child hierarchy

The SAS code for this hierarchy/tree structure follows:

```
data HIERARCHY;
input Parent Child;
datalines;
. 1
1 2
1 3
1 8
1 9
2 4
3 5
5 6
6 7
8 10
10 11
10 13
;
run;
```

APPLYING PROC TREE

A hierarchy with unique descending kinships, i.e. without any cycles, as exemplified in the dataset HIERARCHY, can be visualized with PROC TREE. In our case, we do not know à priori how many parent-child combinations or how many generations of children there are – we need SAS to help us out.

The following syntax extracts the lineage from the example dataset:

```
ods listing;
proc tree data = HIERARCHY prune = 10000 list;
          parent PARENT;
          name CHILD;
          ods output treelisting = GRAPHDATA;
run;
```

In this syntax, the example information is entered in CAPITAL letters: HIERARCHY, 10000, PARENT, CHILD and GRAPHDATA.

A few things should be noted when using PROC TREE. Either the PRUNE option or the equivalent LEVEL option must be set to a high value so that the entire tree height is covered – all nodes between the root and the leaves must be included in the flattening. The ODS LISTING system option must be activated before running the procedure and the LIST option is needed so that the tree structure is also

directed to ODS table TREELISTING in the ODS OUTPUT statement. Should not any dataset name be assigned to TREELISTING, SAS applies its naming standard Data*1*, Data*2*,…, Data*n* depending on what is available in the work folder.

Should there be nodes, Parent and Child combinations, that are not rooted anywhere such that they do not have a leading null value as Parent on their top node, then a warning message is printed from PROC TREE, for example: `WARNING: Parent 18 of 19 not found.`

ODS OUTPUT TREELISTING: WHAT IT CONTAINS

In the ODS OUTPUT, the dataset from TREELISTING contains two types of variables: *h* and *d*. As seen from the VIEWTABLE GRAPHDATA in Display 1, the type *h* appears to be column headings and type *d* appears to be the actual data seen in the SAS OUTPUT window in Display 2. The *batch* column contains the procedure name and the data that make up the tree: CHILD, Height, Parent and Children.

Display 1 shows ODS OUTPUT TREELISTING = GRAPHDATA:

VIEWTABLE: WORK.GRAPHDATA

	type	batch
1	h	The TREE Procedure
2	h	
3	h	CHILD Height Parent Children
4	h	
5	d	1 0 2 3 8 9
6	d	2 1 1 4
7	d	4 2 2
8	d	3 1 1 5
9	d	5 2 3 6
10	d	6 3 5 7
11	d	7 4 6
12	d	8 1 1 10
13	d	10 2 8 11 13
14	d	11 3 10
15	d	13 3 10
16	d	9 1 1

Display 1. ODS TREELISTING dataset from PROC TREE

The output from PROC TREE, given in the SAS OUTPUT window, contains the respective path length (tree depth) and the lineage of every descending branch of the tree, i.e. the unique relationships between the children and their parent nodes, starting at level 0 which is the tree root/top node 1 in the example dataset.

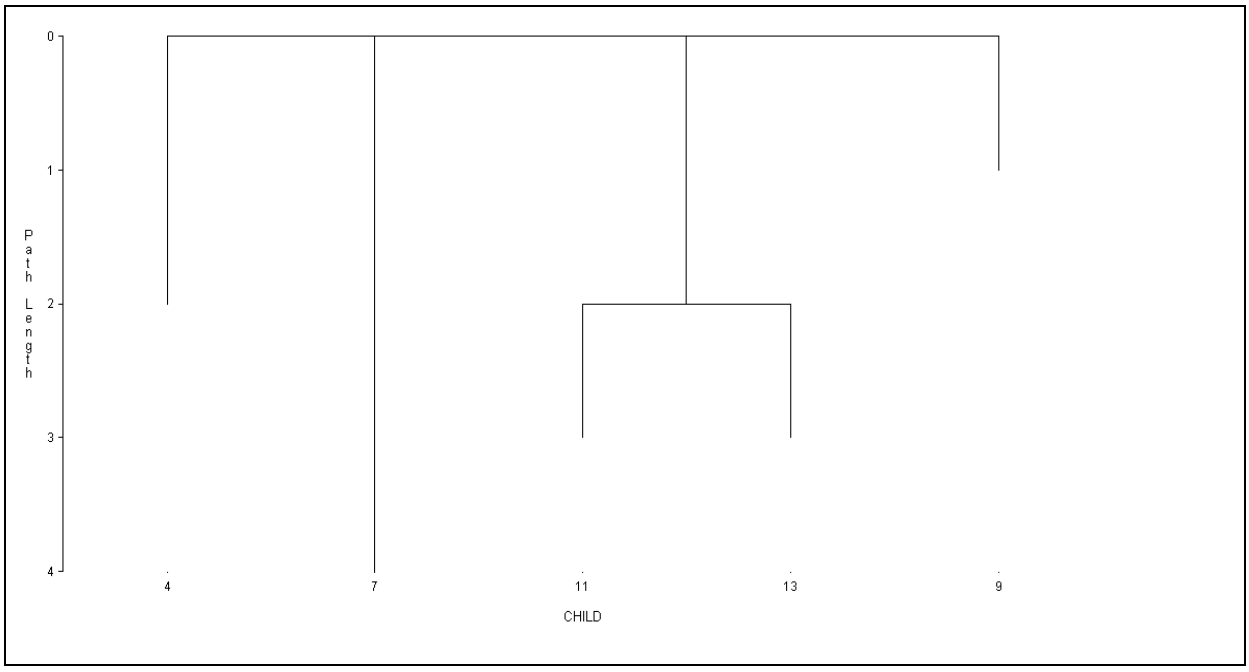
Display 2 contains the SAS OUTPUT from PROC TREE.

The TREE Procedure

CHILD	Height	Parent	Children			
1	0		2	3	8	9
2	1	1	4			
4	2	2				
3	1	1	5			
5	2	3	6			
6	3	5	7			
7	4	6				
8	1	1	10			
10	2	8	11	13		
11	3	10				
13	3	10				
9	1	1				

Display 2. SAS OUTPUT from PROC TREE

Display 3 contains the Graph from PROC TREE.



Display 3. Graph created by PROC TREE

EXTRACTING THE *PATH LENGTH* FROM THE ODS OUTPUT DATASET

The *path length*, or the height of the tree at each node, is seen in the *Height* section of the *batch* column in the ODS output dataset. With some simple handcrafting, the maximum height, or the maximum value for path length as shown in the leftmost side in the SAS OUTPUT, is obtained with this code:

```
data PATH_LENGTH;
 set GraphData (where=(type='d')) end=FINALE;

 Path_Length1 = index(Batch,'');
 Path_Length2 = left(substr(Batch,Path_Length1));
 Path_Length3 = index(Path_Length2,'');
 Path_Length4 = compress(substr(Path_Length2,1,Path_Length3-1));
 Path_Length5 = input(Path_Length4,best12.);

 Retain Path_Length;
 if Path_Length5>Path_Length then Path_Length=Path_Length5;
 if FINALE;
 Call SYMPUTX('PATH_LENGTH',Path_Length);
run;
```

FLATTENING OUT THE HIERARCHY DATASET AND HOW TO USE THE PATH LENGTH

A hierarchy can be flattened out column-wise into a dataset that will contain the lineage of each parent to all generations of children. As a result, each level of descendants is stacked as a column instead of one long sequence in the input data. One way of flattening out the data is according to the following steps.

Initially, parent-child relationships are stacked in two columns, as in the example dataset HIERARCHY. Children that also are parents will be occurring “down the road”, so remerging the dataset with itself appears as a viable option as long as the variable names can be interlinked. This will be similar to a repetitive self-join until the tree depth, as given by the path length variable, is reached.

Step 1. Multiply the original dataset by self-merging it into new augmentation datasets, with linkable variable names

Multiply the hierarchy dataset with itself so that the Parent and Child columns are renamed pairwise, for instance as [C1,C2], [C2,C3],…,[C(i),C(i+1)] in each new augmentation dataset PC(i). This is repeated as many times as the tree depth/path length after the initial root node.

Step 2. Remerge the multiplied augmentation datasets with each other to get the complete lineage
Once all multiples of the original dataset are created and variables are renamed increasingly to allow for matching, the principle is straightforward:

2.1 Sort the first two augmentation datasets by their common linking variable.

2.2 Merge the two augmentation datasets through the common linking variable. This is now a bucket dataset.

2.3 Repeat steps 2.1 and 2.2 by using the previously merged augmentation dataset, which now will contain all linking variables in columns, hence the name “bucket dataset”.

2.4 Keep merely the lineage associated with the top node. This will peel off redundancy matching.

AN EXAMPLE MACRO TO CREATE THE AUGMENTATION DATASETS THROUGH SELF JOIN

In the following macro, the example dataset HIERARCHY is joined with itself in PROC SQL into a new dataset PC(i) for each augmentation and the Parent and Child variables are renamed with an increasing index suffix:

```
%macro MULTIPLY;
   %do i=1 %to &PATH_LENGTH;
   proc sql;
   create table PC&i as

         select
                P.CHILD as  Child&i,
                C.CHILD as  Child%eval(&i+1)

         from (HIERARCHY P, HIERARCHY C)
         where (P.CHILD = C.PARENT)
         ;
   quit;
   %end;
%mend;
%MULTIPLY;
```

Seen in the SQL join in the macro *MULTIPLY*, the HIERARCHY dataset is joined with itself so that the CHILD column is matched with the PARENT column from right. Since the top node had a leading null, it will not be matched, whereas every other node that has a parent will be put in a column named CHILD(i), and the former CHILD column will be named CHILD(i+1). This is done repeatedly so that there are as many multiples of the same dataset as the tree depth, with one increasing variable name in each augmentation: [CHILD1,CHILD2], [CHILD2,CHILD3],[CHILD3,CHILD4],…,[CHILD(i),CHILD(i+1)].
The loop indexing goes up to the path length but the final child is CHILD(i+1) since the hierarchy/tree started at level zero, as seen in Display 3.

Display 4 shows an augmentation of one dataset into four multiples, sufficient for the example dataset HIERARCHY.

$$\begin{bmatrix} . & a_{11} \\ a_{11} & a_{12} \\ a_{12} & a_{13} \\ a_{13} & a_{14} \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \quad \begin{bmatrix} . & a_{11} \\ a_{11} & a_{12} \\ a_{12} & a_{13} \\ a_{13} & a_{14} \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \quad \begin{bmatrix} . & a_{11} \\ a_{11} & a_{12} \\ a_{12} & a_{13} \\ a_{13} & a_{14} \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \quad \begin{bmatrix} . & a_{11} \\ a_{11} & a_{12} \\ a_{12} & a_{13} \\ a_{13} & a_{14} \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}$$

Display 4. Dataset augmentation: multiples of one dataset placed side-by-side

A MERGING SEQUENCE FOR OBTAINING THE FINAL BUCKET DATASET WITH ALL LINEAGES

Once the dataset augmentation has been accomplished, all new datasets can be merged into one final bucket dataset. The principle is that the first two augmentation datasets are merged on their common variable into one combined dataset, now referred to as a bucket dataset. Then, the next augmentation dataset it merged with the merge of the two previous (now the bucket dataset) on the common variable from the second augmentation dataset that is in the bucket. This is repeated until all generations of children are covered when the final augmentation dataset is merged. The final bucket dataset will then contain all lineages.

AN EXAMPLE MERGING SEQUENCE FOR OBTAINING THE FINAL BUCKET DATASET

In the following syntax, the augmentation datasets are matched one by one from the right hand side. At each merge, the obtained dataset is named Bucket(i), and for simplification, the first augmentation dataset PC1 is renamed to Bucket0 before the first matching. After that, each merge is between the just obtained new bucket and the next augmentation dataset PC(i+1), which has one common variable with the bucket, CHILD(i+1). Also, the top node or tree root is identified from the HIERARCHY dataset since it is not in the augmentation datasets (it could not be self-joined) and placed in a macro variable &TOP_NODE. Then the merging sequence starts:

```
data _NULL_;
   set HIERARCHY;
   IF PARENT=.;
   CALL SYMPUTX('TOP_NODE',CHILD);
run;

proc datasets nolist library=work;
     delete BUCKET0;
     change PC1      = BUCKET0;
quit;

proc sort data=BUCKET0; by CHILD2; run;
proc sort data=PC2;     by CHILD2; run;

data BUCKET1;
     merge BUCKET0  (IN=MUSTBE)
           PC2      (IN=OPTIONAL)
           ;
     by CHILD2;
     if MUSTBE;
run;

proc sort data=BUCKET1; by CHILD3; run;
proc sort data=PC3;     by CHILD3; run;

data BUCKET2;
     merge BUCKET1  (IN=MUSTBE)
           PC3      (IN=OPTIONAL)
           ;
     by CHILD3;
     if MUSTBE;
run;

proc sort data=BUCKET2; by CHILD4; run;
proc sort data=PC4;     by CHILD4; run;

data BUCKET3;
     merge BUCKET2  (IN=MUSTBE)
           PC4      (IN=OPTIONAL)
           ;
     by CHILD4;
     if MUSTBE;
run;
```

```
Data DELINEAGE;
      SET BUCKET%eval(&PATH_LENGTH-1);
      if CHILD1=&TOP_NODE;
run;
```

The repeated merging sequence can also be written as a macro loop, after having renamed the first augmentation dataset PC1 as the initial bucket, Bucket0:

```
%do i=1 %to %eval(&Path_Length-1);
proc sort data=BUCKET%eval(&i-1); by CHILD%eval(&i+1); run;
proc sort data=PC%eval(&i+1);     by CHILD%eval(&i+1); run;

data BUCKET%eval(&i);
     merge   BUCKET%eval(&i-1)   (IN=MUSTBE)
             PC%eval(&i+1)       (IN=OPTIONAL)
             ;
     by CHILD%eval(&i+1);
     if MUSTBE;
run;
%end;
```

Obtained in the final step, the DELINEAGE dataset contains the complete lineage in HIERARCHY, seen in Display 5:

VIEWTABLE: WORK.DELINEAGE

	Child1	Child2	Child3	Child4	Child5
1	1	9	.	.	.
2	1	2	4	.	.
3	1	3	5	6	7
4	1	8	10	11	.
5	1	8	10	13	.

Display 5. The final de-lineage of dataset Hierarchy

A SOLUTION PROVIDED BY SAS: SAMPLE 25968 IN THE SAS KNOWLEDGE BASE

An alternative way of obtaining parent child de-lineage is to apply the sample algorithm provided by SAS® in the SAS Knowledge Base, sample 25968. This is a solution based on just one data step and the use of the POINT= option in the data step. Also, it is supplied with pedagogical comments.

The sample is here reprinted with the hierarchy dataset TEST, corresponding to the HIERARCHY dataset used previously in this paper. As both datasets, TEST and HIERARCHY contain PARENT and CHILD variables, the sample code can be applied directly to HIERARCHY by simply replacing the two occurrences of the dataset TEST, and since the depth is 4, no changes are necessary in the number of columns needed:

```
data lineage(keep=hist1-hist5);

/* Read in SAS data set */
```

```
set TEST;

/* Create an array with enough elements to hold the maximum number of
   observations that will 'link' together. */

array hist(5);
count=1;

/* Put the value of CHILD variable into the array HIST */
hist(count)=child;

/* Process through the entire data set while the value of COUNT is less
   than 4, i.e. the  upper boundary of the array minus 1.  */

do i=1 to last while (count<dim(hist)-1);

/* Read in SAS data set again.  Rename the variables to new names so
   they can be compared with the variable names coming in with the
   first SET statement. The POINT= option allows you to access each
   observation by observation number.*/

set TEST(rename=(child=child1 parent=parent1)) nobs=last point=i;

/* As you step through each observation, compare the value of PARENT to
   the value of CHILD1 (which is the new name given to the CHILD variable
   when data set is brought in second time). */

if parent=child1 then do;
   count+1;
   /* populate array with value of CHILD1 */
   hist(count)=child1;
   child=child1;
   parent=parent1;
   i=1;
 end;
end;
/* increment COUNT */
count+1;
hist(count)=parent;
run;
```

CONCLUSION

The use of PROC TREE simplifies the effort needed to obtain the lineage in hierarchy data/tree data. Even though there is some programming left to do once the procedure is applied, it provides an excellent shortcut to the flattened structure, de-lineage, as well as it offers a verification opportunity through its output data and graph.

To summarize, PROC TREE is most likely not the only SAS® procedure or function that embeds hidden functionality and provides versatile usage: with the ODS output, the tree height/path length can be extracted and a parent child de-lineage or hierarchy/tree flattening out can be achieved without profound programming skills.

REFERENCES

Koski, T. and Nobel, J. 2009. "Bayesian Networks *An Introduction"*. Wiley.

Nizol, M. 2010. "Tables as Trees: Merging with Wildcards using Tree Traversal and Pruning." *MidWestSASUsersGroup (MWSUG) Proceedings,* paper 49-2010, Milwaukee, Wisconsin, USA.

SAS Knowledge Base, Sample 25968, "Using POINT= to trace 'parent-child' lineage in a data set". Retrieved on March 8th, 2016 from

http://support.sas.com/kb/25/968.html

ACKNOWLEDGMENTS

The Swedish SAS® Support team, specifically Mrs. Lena Norberg and Mr. Georgios Karagiannis, have contributed with valuable insights during this work. I gratefully acknowledge their input.

CONTACT INFORMATION

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

Can Tongur
Statistics Sweden
Can.Tongur(at)scb.se
www.scb.se