PharmaSUG 2017 – Paper AD11

Generating Homeomorphically Irreducible Trees (Solving the Blackboard Problem in the Movie "Good Will Hunting")

John R Gerlach, Dataceutics, Inc., Pottstown, PA USA

ABSTRACT

In the movie, *Good Will Hunting* (1997), a mathematics professor challenges his students to draw all Homeomorphically Irreducible Trees of Order Ten, that is, a collection of trees each having ten dots connected by lines. The well-known blackboard problem in the movie poses a formidable challenge, especially for larger trees having twenty or thirty nodes. It would require an extremely large blackboard to draw all the trees, as well as to erase those deemed redundant or incorrect. This paper explains a SAS® solution for generating Homeomorphically Irreducible Trees of order *N*.

INTRODUCTION

Table 1 shows the complete list of Homeomorphically Irreducible Trees (from hereon called HMI-Trees) of Order Ten (N=10), such that each tree contains ten dots connected by nine lines. Observe the number of external and internal nodes for each tree; and note that there are *least three lines* originating from an internal node.

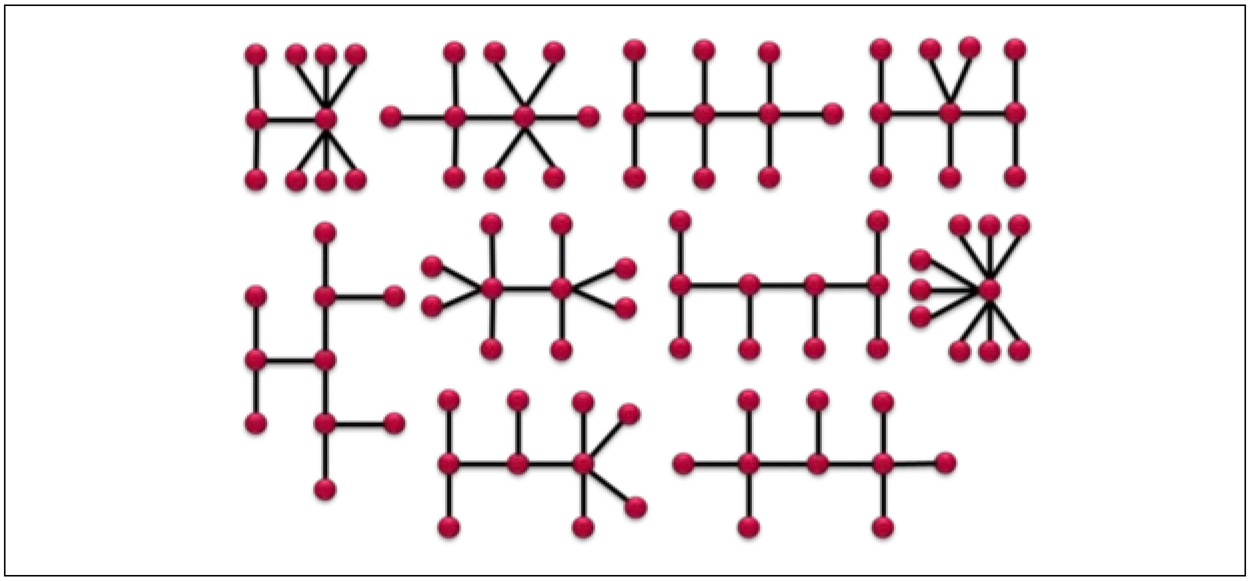
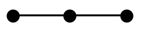
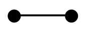


Table 1: Set of HMI-Trees having 10 nodes.

A *Homeomorphically Irreducible Tree*: quite a mouthful. But what is it? For that matter, what is a tree? Well, for this discussion, a tree is a network of dots (nodes) and line segments. There are no cycles allowed; that is, a line cannot return from where it originated. Homeomorphism indicates that the same tree can be drawn differently, but are topologically equivalent. For example, each tree in Table 1 could be written as a mirror image of itself, but would be considered the same tree. Irreducible demands that a vertex (internal node) must have *more than two lines* drawn from it, as shown below, since the objective is to draw *unique* trees having N-nodes.

 *is actually*  *having two external nodes*

The proposed SAS solution includes a notational convention {i_1, i_2, i_3, ...,, i_i} that defines a unique tree where i_i denotes an internal node having degree three, at least. The convention is based on the idea that a tree is *partitioned with respect to its internal nodes*. In other words, contrary to the *irreducible* component of HMI-Trees, the notation distinguishes each *internal* node as *a separate entity*. For example, consider the HMI-Tree {4,3} shown in Table 2. Imagine separating the first internal node of degree four from the second internal node of degree three, albeit each sharing a common line segment. In fact, the crux of the proposed SAS solution is based on the notational scheme and the partitioning of the tree with respect to internal nodes.

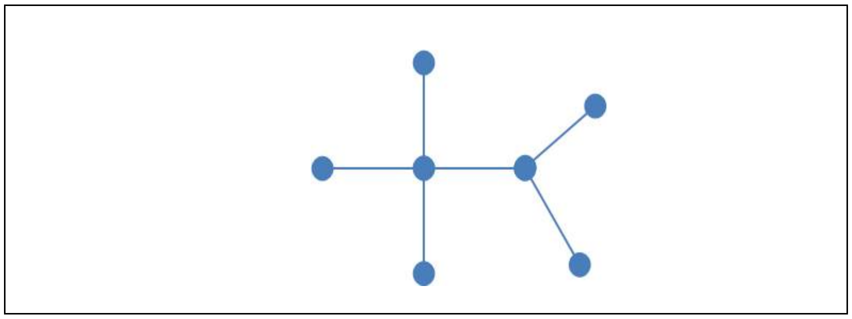


Table 2: HMI-Tree having the notation {4,3}.

As an exercise, the reader should be able to draw the following trees written in notational form: {4}, {3,6}, {3,3,4}, {9}, {4,4,4,5}, {3,3,3,3}. The solution {3,3,3,3} actually represents *two distinct trees*, thereby indicating a caveat of the SAS solution, since it can discern only one tree per notation. The reader should attempt to draw both trees from {3,3,3,3}. Another good exercise – Write the notation for the trees shown in Table 1: {3,7} for starters.

PROPERTIES OF HMI-TREES

Let *N* denote the number of nodes for a set of HMI-Trees. What are the trees? How many are there? How does one begin to draw such trees? Are there patterns as N increases? Yes, there are. Certainly, there is always the case of having one internal code from which there would be *N-1* lines connecting the external nodes.

Consider the following properties of HMI-Trees:

- For N-nodes, there are N-1 lines in a tree.
- There are no HMI-Trees for N=3.
- For N greater than 3, there is always the tree { (N-1) }, one internal node and N-1 external nodes.
- For even-valued N greater than 3, there are [(N-2) / 2] internal nodes of degree value 3.
- Beginning with N=8, there are trees:
    - { 3, {N-5) }
    - { 4, {N-4) }.
- Beginning with N=10, there is the tree { 4, 3, (N-6) }.
- As N increases, the number of trees increases exponentially.

Although these properties give insight to the formulation of trees, it offers little help when trying to implement a programmatic solution.

THE IDEA

The conceptual solution for generating HMI-Trees begins by looking at a tree as *a collection of partitioned components each centered on their respective internal node*. This idea makes the notational scheme intuitive and robust, such as the aforementioned {4,3} tree that has two internal nodes of degree four and three, in that order. Notice that the tree {4,3} and {3,4} are recognized as equivalent, hence one being redundant. However, the process of generating trees requires knowing more than the number of external and internal nodes.

Because a tree is processed as a collection of partitioned components, the common line segments connecting them must be counted *twice*, which inflates the total number of lines in a tree (i.e., the sum of degrees). For example, the tree {4,3} has a common line segment that contributes twice to the total number of lines for that tree, hence 4+3=7, even though there are actually six lines in the tree. In fact, the total number of lines, denoted by the variable TLINES, helps to determine whether a tree is valid.

Concerning HMI-Trees of Order 10, there are (coincidentally) ten trees, for example the tree consisting of nine external nodes (ENODES) and its compliment number of internal nodes (INODES). Moreover, for a given pair of external and internal node values, there is a respective number of total lines (TLINES) computed by the formula INODES + (N-2), as shown in Table 3 below, ranging from 9 to 17 for Order 10.

Are all combinations of External-Internal (EI) pairwise values, written as an ordered pair (ENODES, INODES), needed to determine the collection of trees? It might appear that the ordered pair (3,7) produces a similar tree as (7,3); however, a tree with three external nodes hardly looks like one with seven external nodes. In fact, the EI pair (3,7) is not even an HMI-Tree! It turns out that those highlighted rows in yellow are extraneous combinations that do not offer any solutions. But how do you know that? Well, recall that an internal node must be of degree three, *at least*. So, for example, given a tree having six internal nodes, then there must be *at least eighteen lines* (3*6=18), as indicated by the first arrow in Table 3. However, by the aforementioned formula, there can be *only* fourteen lines for a tree having four external nodes and six internal nodes. Therefore, this EI pairwise combination can be discarded outright because it does not meet the *minimum threshold* needed for defining a tree, represented by the column MLINES in Table 3.

		===== # Nodes ====		======= # Lines Per Component =====		
	OBS	**ENODES**	**INODES**	**TLINES**	**MLINES**	
		(External)	**(Internal)**	***INODES + (N-2)***	***(3 * INODES)***	
Order 10	1	9	1	1 + (10-2) = 9	3* 1 = 3	
	2	8	2	2 + (10-2) = 10	3* 2 = 6	
	3	7	3	3 + (10-2) = 11	3* 3 = 9	
	4	6	4	4 + (10-2) = 12	3* 4 = 12	
	5	5	5	5 + (10-2) = 13	3* 5 = 15	
	6	4	6	6 + (10-2) = 14	3* 6 = 18	←
	7	3	7	15	21	
	8	2	8	16	24	
	9	1	9	9 + (10-2) = 17	3* 9 = 27	
Order 11	1	10	1	1 + (11-2) = 10	3* 1 = 3	
	2	9	2	2 + (11-2) = 11	3* 2 = 6	
	3	8	3	3 + (11-2) = 12	3* 3 = 9	
	4	7	4	4 + (11-2) = 13	3* 4 = 12	
	5	6	5	5 + (11-2) = 14	3* 5 = 15	←
	6	5	6	15	18	
	7	4	7	16	21	
	8	3	8	17	24	
	9	2	9	18	27	
	10	1	10	19	30	

Table 3: Combinations of External-Internal Nodes for Order 10, 11 and 12.

Just for the record, it is incorrect to consider only those combinations based on the rule that the number of external nodes exceeds the number of internal nodes, for example, the disqualified EI ordered pair (6,5), indicated by the second arrow, found in Order 11.

The data mart needed for generating well-defined trees needs one more piece of information -- a vector of values denoting all the possible degree-values for Order *N*. Consider the heuristic SAS code in Table 4 that creates the data mart needed for Order 10. The %LET statement defines the only parameter required for the proposed SAS solution, which denotes the number of nodes, the Order of HMIT-trees. Because internal nodes must be of degree three, at least, and no more than N-1, the macro %degree enumerates the values of the vector from 3 to N-1 defined in an ARRAY statement in the following Data step. The DO-loop decrements the number of external nodes from N-1 to 1 and produces the following variables:

• ENODES	# External Nodes	DO-Loop Control Variable
• INODES	# Internal Nodes	N - ENODES
• TLINES	# Lines in Tree	INODES + (N – 2)
• MLINES	# Lines in Tree (Minimum)	3 * INODES

```
%let n = 10;

%macro degrees;
   %do i = 3 %to %eval(&n.-1); &i. %end;
%mend degrees;

data datamart;
   retain n &n.;
   array vertex{%eval(&n.-1)} v3-v%eval(&n.-1) (%degrees);
   do enodes = %eval(&n.-1) to 1 by -1;
      inodes = n - enodes;
      tlines = inodes + (&n.-2);
      mlines = 3 * inodes;
      if tlines ge mlines
         then output;
      end;
run;
```

Table 4: Heuristic SAS code to generate the data mart for generating HMI-Trees.

Table 5 shows the data mart needed to generate viable HMI-Trees of Order 10. Thanks to the lesson learned concerning the total lines (TLINES) meeting a minimum threshold (MLINES), the Data step produces only viable combinations of external / internal nodes that will produce candidate trees.

	==== # Nodes ====		=== # Lines ===		====== Degree Value ====== (# Lines from Node)							
Order	**External**	**Internal**	**Total**	**Minimum**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
N	**ENODE**	**INODE**	**TLINES**	**MLINES**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**	
10	9	1	9	3	3	4	5	6	7	8	9	***Base Case***
10	8	2	10	6	3	4	5	6	7	8	9	←
10	7	3	11	9	3	4	5	6	7	8	9	
10	6	4	12	12	3	4	5	6	7	8	9	←

Table 5: Data mart for generating HMIT-trees of Order 10.

First consider the Base Case, that is, one internal node and *N-1* external nodes, for which there is only one possible degree value: {9}. Next, consider the pairwise External-Internal ordered pair (8,2), indicated by the first arrow in Table 5. Because the degree-values must add up to the respective value of TLINES (i.e. 10), there are only several possibilities: {3,7} and {4,6}. Are there any other trees? Of course. Moving downward in Table 5, consider the pair-wise nodes (6,4), as indicated by the second arrow. Are there four internal nodes whose degree-values add up to twelve? Yes, the tree {3,3,3,3} is valid, which becomes an issue (a caveat) that will be discussed later.

Although the heuristic code in Table 4 helps to understand how a tree is derived, the actual SAS solution utilizes nested DO-loops such that the loop control variables provide all the possible combinations of degree values. In summary, the vector of degree-values consist of constant values germane to a given Order; whereas, the other variables (e.g. TLINES) are always a function of the pairwise values of external / internal nodes.

THE SAS SOLUTION

There is only one user-specified parameter *N*, which denotes the Order of HMI-Trees, as shown below for Order 10. Otherwise, the proposed solution uses SAS Base primarily consisting of nested DO-loops that are utilized, accordingly, depending on the number of external nodes.

```
 %let N = 10;
```

The following Data step generates the data mart as a function of the number of external nodes such that the **first** iteration of the main DO-loop represents *N-1* external nodes, the so-called Base Case, from which the other data points are computed: INODES, TLINES, and MLINES. If the value of TLINES meets the minimum threshold of making a tree (i.e. MLINES), the process proceeds, otherwise it exits the main loop and the Data step terminates.

```
data ds01;
    length _numeric_ 3;
    do enodes = %eval(&N.-1) to 1 by -1;
       inodes = &N. - enodes;         * Number of internal nodes   ;
       tlines = inodes + (&N.-2);     * Total lines required       ;
       mlines = 3 * inodes;           * Minimum lines required     ;
       if tlines ge mlines
          then ;
          else leave;                 * Test for Minimum Threshold ;
```

For the first iteration of Order 10, the values of the variables are:

•	ENODES	→	9	DO-Loop Control Variable
•	INODES	→	1	N - ENODES
•	TLINES	→	9	INODES + (N – 2)
•	MLINES	→	3	3 * INODES.

Because the total lines is greater than the minimum threshold required, the process proceeds to select the appropriate condition of the SELECT/WHEN statement. There is only one internal node; therefore, the first WHEN clause contains only one DO-loop whose loop control variable (INODE1) denotes the degree value. Notice that the DO-loop iterates from 3 to the number of external nodes. The IF statement within the DO loop tests whether the degree-value equals TLINES, which identifies a tree. Consequently, the Base Case having notation {9} is obtained and control returns back to the main loop to re-generate the data mart representing eight external nodes.

```
       select(enodes);
          when(%eval(&N.-1)) do;
             do inode1 = 3 to enodes;
                if inode1 eq tlines
                   then output;
                end;
             end;
```

The number of internal nodes increases by one as the number of external nodes decrements by one; thus, there needs to be an additional DO-loop in order to consider combinations of degree-values. Accordingly, for seven external nodes of Order Ten, there needs to be three DO-loops, representing the internal nodes, thereby generating trees having the notation {x_1, x_2, x_3}, such as: {3,4,4}. Note: The reader should be able to draw this tree.

The remaining SAS code displays enough WHEN clauses for Order 10 and indicates where more clauses would be included, as needed.

```
            when(%eval(&N.-2)) do;
               do inode1 = 3 to enodes;
                  do inode2 = 3 to enodes;
                     if (inode1 + inode2) eq tlines
                        then output;
                     end;
                  end;
               end;
            when(%eval(&n.-3)) do;
               do inode1 = 3 to enodes;
                  do inode2 = 3 to enodes;
                     do inode3 = 3 to enodes;
                        if (inode1 + inode2 + inode3) eq tlines
                           then output;
                        end;
                     end;
                  end;
               end;
            when(%eval(&n.-4)) do;
               do inode1 = 3 to enodes;
                  do inode2 = 3 to enodes;
                     do inode3 = 3 to enodes;
                       do inode4 = 3 to enodes;
                           if (inode1 + inode2 + inode3 + inode4) eq tlines
                              then output;
                           end;
                        end;
                     end;
                  end;
               end;

       * More WHEN clauses needed, accordingly  ;;

             otherwise;
             end;
         end;
      drop inode: enodes;
   run;
```

Notice that each WHEN clause represents the number of external nodes while the nested DO loops inside the clause represent the number of internal nodes, which produce the combinations of degree values. For larger *N*, there needs to be many WHEN clauses and even more nested DO-loops. How many should there be? Well, it depends on the Order *N* being analyzed, obviously. More importantly, the SAS program becomes extremely long and tedious for larger *N*. A solution to this problem using the Macro Language will be discussed later.

The nested-iterative process generates a collection of trees, called *candidates*, some of which are redundant. Consider the candidate trees: {3,4,4}, {4,3,4}, {4,4,3} of Order 10, one of which is superfluous. But which one? How do you discern similar trees that have a different notational sequence? Good question. It so happens that these trees have the same number of instances of degree-values (1 of degree 3, 2 of degree 4), thereby identifying them as a group. So, let's define a pattern variable, called GROUP, denoting the number of instances of each degree-value for these several trees, as follows:

```
010203040506070809  ← Degree-Values for Order 10
000001020000000000  ← Pattern Variable for {3,4,4},{4,3,4},{4,4,3}
```

That's a good start, but there needs to be two more pattern variables: one denoting the actual tree and the other being its reverse. Thus, the tree {3,4,4} would have the patterns: 030404 and its reverse 040403. The following Data step creates these *very important* pattern variables (GROUP, ACTUAL, REVERSE), which are used to sort the resultant data set.

Table 6 shows HMI-Trees for Order 10 organized in groups denoting the instances of degrees, *which make them simila*r. However, the table reveals a more interesting facet about these similar trees with respect to the other pattern variables: ACTUAL and REVERSE. Notice that there are instances where the pairwise values of the pattern variables are exact mirror images of each other. Therefore, the similar trees trees {3,4,4} and {4,4,3} are topologically the same. Notice that the tree {4,3,4} differs from the other two in the same group, hence, it is deemed unique. Of course, one of the two similar trees must be discarded. However, once again it begs the question: Which one?

```
data ds02;
   length group actual reverse $500;
   array tdegrees{*} tdegree1-tdegree&n.;
   array inode{*}  inode1-inode&n.;
   set ds01;
   do i = 1 to %eval(&n.-1);
      tdegrees{i} = 0;
      end;
   do i = 1 to %eval(&n.-1);
      if inode{i} ne .
         then tdegrees{inode{i}} = tdegrees{inode{i}} + 1;
      end;
   do i = 1 to %eval(&n.-1);
      group = trim(group) || put(tdegrees{i},z2.);
      if inode{i} ne .
         then actual = compress(trim(actual) || put(inode{i},z2.));
      end;
   do j = %eval(&n.-1) to 1 by -1;
      if inode{j} ne .
         then reverse = compress(trim(reverse) || put(inode{j},z2.));
      end;
run;

proc sort data=ds02 out=ds03;
   by group actual reverse;
run;
```

```
GROUP              ACTUAL   REVERSE   ENODES INODES  INODE1 INODE2 INODE3 ONODE4

000000000000000001 09       09            9      1      9      .      .      .

000000000200000000 0505     0505          8      2      5      5      .      .

000000010001000000 0406     0604          8      2      4      6      .      .
                   0604     0406          8      2      6      4      .      .

000001000000010000 0307     0703          8      2      3      7      .      .
                   0703     0307          8      2      7      3      .      .

000001020000000000 030404   040403        7      3      3      4      4      .
                   040304   040304        7      3      4      3      4      .
                   040403   030404        7      3      4      4      3      .

000002000100000000 030305   050303        7      3      3      3      5      .
                   030503   030503        7      3      3      5      3      .
                   050303   030305        7      3      5      3      3      .

000004000000000000 03030303 03030303      6      4      3      3      3      3
```

Table 6: The Pattern variables for Order 10 in sort-order.

The next and final Data step performs the task of selecting unique trees (i.e. discarding redundant trees) by converting the pattern variables ACTUAL and REVERSE into numeric values, then selecting only those instances where the first pattern *is less than or equal to* the second pattern. For example, consider the trees {3,4,4} and {4,4,3} whose ACTUAL and REVERSE values are *reciprocal versions of each other*, thereby indicating a topological equivalence, then selecting the first one by numerical comparison. Finally, the same Data step creates a variable called TREE that represents the unique HMI-Tree, in notational form, by parsing the variable ACTUAL, accordingly.

```
data ds04;
   length tree $500;
   set ds03;
   if (input(patt1,best.) le input(patt2,best.))
      then do;
         vals = length(trim(left(actual)));
         do i = 1 to vals by 2;
            if i+1 lt vals and vals ne 2
               then tree = trim(tree) || substr(patt1,i,2) || ',';
               else tree = trim(tree) || substr(patt1,i,2);
            end;
         tree = '{' || trim(left(tree)) || '}';
         output;
         end;
run;
```

Table 7 shows HMI-Trees for Order 10 organized by the pattern variables, including the unique trees, written in notation form. Notice that there are nine trees even though there are supposed to be ten HMI-Trees of Order 10. What happened? It turns out that the tree {3,3,3,3} can be drawn two different ways. Unfortunately, the SAS solution discerns only one tree per notation. Nonetheless, the SAS solution generates a precise minimum number of trees for Order *N*. Notice that the total of the degree value(s) add up to the value of TLINES, the total number of lines, which is always less than MLINES (3 times the number of internal nodes). It is that criterion that selects the viable set, then an IF-statement selects the numeric version of Patterns 1 and 2, specifically that the first (pattern) value is less than or equal to the second one, producing the list of unique HMI-Trees.

```
      GROUP               PATT1        PATT2         ENODES     INODES     TREE

 000000000000000001      09           09               9          1      {09}

 000000000200000000      0505         0505             8          2      {05,05}

 000000010001000000      0406         0604             8          2      {04,06}

 000001000000010000      0307         0703             8          2      {03,07}

 000001020000000000      030404       040403           7          3      {03,04,04}
                         040304       040304           7          3      {04,03,04}

 000002000100000000      030305       050303           7          3      {03,03,05}
                         030503       030503           7          3      {03,05,03}

 000004000000000000      03030303     03030303         6          4      {03,03,03,03}
```

Table 7: HMI-Trees of Order 10. Notice that there are only nine, not ten trees.

HMI-TREE SOLUTIONS

Consider the more obvious solutions for Order 1,2,4,5 and 6 (recall, there is no solution for Order 3). Order 1 generates a single node and Order 2 generates a line segment having to external nodes. Order 4 and Order 5 each generate one tree that represents the aforementioned base case {*N*-1}, that is, having one internal node connecting he remaining external nodes. The situation gets more interesting with Order 6, which generates two trees: {5} and {3,3}. From then on, the problem becomes far more challenging. There are four trees for Order 7: {6}, {2,4,4}, {2,3,5} and {5,3,3,3,}. There are four trees for Order 8 and five trees for Order 9. Peruse the following list of trees for Orders 10 through 16. Notice the highlighted trees having a *constant degree* value, such as the trees {5,5} and {3,3,3,3} found in Order 10.

Order 10: 9 Trees (Note: There are actually 10 unique trees)

{9} {3,7} {4,6} {5,5} {3,3,5} {3,4,4} {3,5,3} {4,3,4} {3,3,3,3}

Order 11: 12 Trees

{10} {3,8} {4,7} {5,6} {3,3,6} {3,4,5} {3,5,4} {3,6,3} {4,3,5} {4,4,4}
{3,3,3,4} {3,3,4,3}

Order 12: 21 Trees

{11} {3,9} {4,8} {5,7} {6,6} {3,3,7} {3,4,6} {3,5,5} {3,6,4} {3,7,3} {4,3,6} {4,4,5}
{4,5,4} {5,3,5} {3,3,3,5} {3,3,4,4} {3,3,5,3} {3,4,3,4} {3,4,4,3} {4,3,3,4}
{3,3,3,3,3}

Order 13: 30 Trees

{12} {3,10} {4,9} {5,8} {6,7} {3,3,8} {3,4,7} {3,5,6} {3,6,5} {3,7,4} {3,8,3}
{4,3,7} {4,4,6} {4,5,5} {4,6,4} {5,3,6} {5,4,5} {3,3,3,6} {3,3,4,5} {3,3,5,4}
{3,3,6,3} {3,4,3,5} {3,4,4,4} {3,4,5,3} {3,5,3,4} {4,3,3,5} {4,3,4,4} {3,3,3,3,4}
{3,3,3,4,3} {3,3,4,3,3}

Order 14: 51 Trees

{13} {3,11} {4,10} {5,9} {6,8} {7,7} {3,3,9} {3,4,8} {3,5,7} {3,6,6} {3,7,5} {3,8,4}
{3,9,3} {4,3,8} {4,4,7} {4,5,6} {4,6,5} {4,7,4} {5,3,7} {5,4,6} {5,5,5} {6,3,6}
{3,3,3,7} {3,3,4,6} {3,3,5,5} {3,3,6,4} {3,3,7,3} {3,4,3,6} {3,4,4,5} {3,4,5,4}
{3,4,6,3} {3,5,3,5} {3,5,4,4} {3,5,5,3} {3,6,3,4} {4,3,3,6} {4,3,4,5} {4,3,5,4}
{4,4,3,5} {4,4,4,4} {5,3,3,5} {3,3,3,3,5} {3,3,3,4,4} {3,3,3,5,3} {3,3,4,3,4}
{3,3,4,4,3} {3,3,5,3,3} {3,4,3,3,4} {3,4,3,4,3} {4,3,3,3,4} {3,3,3,3,3,3}

Order 15: 76 Trees

{14} {7,8} {6,9} {5,10} {5,5,6} {5,6,5} {4,11} {4,6,6} {6,4,6} {4,5,7}{4,7,5} {5,4,7}
{4,4,8} {4,8,4} {4,4,4,5} {4,4,5,4} {3,12} {3,6,7} {3,7,6} {6,3,7} {3,5,8} {3,8,5}
{5,3,8} {3,4,9} {3,9,4} {4,3,9} {3,4,5,5} {3,5,4,5} {3,5,5,4} {4,3,5,5} {4,5,3,5}
{5,3,4,5} {3,4,4,6} {3,4,6,4} {3,6,4,4} {4,3,4,6} {4,3,6,4} {4,4,3,6} {3,3,10}
{3,10,3} {3,3,5,6} {3,3,6,5} {3,5,3,6} {3,5,6,3} {3,6,3,5} {5,3,3,6} {3,3,4,7}
{3,3,7,4} {3,4,3,7} {3,4,7,3} {3,7,3,4} {4,3,3,7} {3,3,4,4,4} {3,4,3,4,4} {3,4,4,3,4}
{3,4,4,4,3} {4,3,3,4,4} {4,3,4,3,4} {3,3,3,8} {3,3,8,3} {3,3,3,4,5} {3,3,3,5,4}
{3,3,4,3,5} {3,3,4,5,3} {3,3,5,3,4} {3,3,5,4,3} {3,4,3,3,5} {3,4,3,5,3} {3,5,3,3,4}
{4,3,3,3,5} {3,3,3,3,6} {3,3,3,6,3} {3,3,6,3,3} {3,3,3,3,3,4} {3,3,3,3,4,3}
{3,3,3,4,3,3}

Order 16: 127 Trees

{15} {8,8} {7,9} {6,10} {5,11} {5,6,6} {6,5,6} {5,5,7} {5,7,5} {4,12} {4,6,7}
{4,7,6} {6,4,7} {4,5,8} {4,8,5} {5,4,8} {4,4,9} {4,9,4} {4,4,5,5} {4,5,4,5} {4,5,5,4}
{5,4,4,5} {4,4,4,6} {4,4,6,4} {3,13} {3,7,7} {7,3,7} {3,6,8} {3,8,6} {6,3,8} {3,5,9}
{3,9,5} {5,3,9} {3,5,5,5} {5,3,5,5} {3,4,10} {3,10,4} {4,3,10} {3,4,5,6} {3,4,6,5}
{3,5,4,6} {3,5,6,4} {3,6,4,5} {3,6,5,4} {4,3,5,6} {4,3,6,5} {4,5,3,6} {4,6,3,5}
{5,3,4,6} {5,4,3,6} {3,4,4,7} {3,4,7,4} {3,7,4,4} {4,3,4,7} {4,3,7,4} {4,4,3,7}
{3,4,4,4,4} {4,3,4,4,4} {4,4,3,4,4} {3,3,11} {3,11,3} {3,3,6,6} {3,6,3,6} {3,6,6,3}
{6,3,3,6} {3,3,5,7} {3,3,7,5} {3,5,3,7} {3,5,7,3} {3,7,3,5} {5,3,3,7} {3,3,4,8}
{3,3,8,4} {3,4,3,8} {3,4,8,3} {3,8,3,4} {4,3,3,8} {3,3,4,4,5} {3,3,4,5,4} {3,3,5,4,4}
{3,4,3,4,5} {3,4,3,5,4} {3,4,4,3,5} {3,4,4,5,3} {3,4,5,3,4} {3,4,5,4,3} {3,5,3,4,4}

```
{3,5,4,3,4} {4,3,3,4,5} {4,3,3,5,4} {4,3,4,3,5} {4,3,5,3,4} {4,4,3,3,5} {3,3,3,9}
{3,3,9,3} {3,3,3,5,5} {3,3,5,3,5} {3,3,5,5,3} {3,5,3,3,5} {3,5,3,5,3} {5,3,3,3,5}
{3,3,3,4,6} {3,3,3,6,4} {3,3,4,3,6} {3,3,4,6,3} {3,3,6,3,4} {3,3,6,4,3} {3,4,3,3,6}
{3,4,3,6,3} {3,6,3,3,4} {4,3,3,3,6} {3,3,3,3,7} {3,3,3,7,3} {3,3,7,3,3} {3,3,3,3,4,4}
{3,3,3,4,3,4} {3,3,3,4,4,3} {3,3,4,3,3,4} {3,3,4,3,4,3} {3,3,4,4,3,3} {3,4,3,3,3,4}
{3,4,3,3,4,3} {4,3,3,3,3,4} {3,3,3,3,3,5} {3,3,3,3,5,3} {3,3,3,5,3,3} {3,3,3,3,3,3,3}
```

There are several interesting properties about trees having a constant degree value, specifically, for example, the tree {3,3,3} and the next instance {3,3,3,3} of such trees. Trees of constant degree begin in Order *d+1*, where *d* denotes the degree. For example, the tree {3} begins in Order 4 and {8} begins in Order 9. Table 8 displays the association of a constant degree value, ranging from one to seven instances, to their respective Order, where such trees manifest. For example, the trees {4,4,4,4} and {5,5,5} reside in Order 14, as found in the above listing of trees. Surprisingly, Orders 13 and 15 do not contain any trees having constant degree values. It is left for the reader to fathom why.

Constant Degree	**Order *N* Having *1-5* Constant Degree Values**							**Increments**
	1	***2***	***3***	***4***	***5***	***6***	***7***	
3	4	6	8	10	12	14	16	2
4	5	8	11	14	17	20	23	3
5	6	10	14	18	22	26	30	4
6	7	12	17	22	27	32	37	5
7	8	14	20	26	32	38	44	6
8	9	16	23	30	37	44	51	7
9	10	18	26	34	42	50	58	8
10	11	20	29	38	47	63	72	9

Table 8: Association of trees having constant valued degrees and their Order.

What is the Order of an HMI-Tree having *n*-instances of constant degree value? For example, the trees {3,3,3,3,3,3} and {5,5,5} belong to Order 14, while the tree {6,6,6,6} belongs to Order 22, according to the empirical evidence shown in Table 8. But, what about a tree of *n*-instances? Let's consider why the tree {3,3,3,3} belongs to Order 10. Obviously, there are four internal nodes, each of degree 3. When analyzed as partitioned entities, there are 12 nodes, which is an inflated value, due to the notation scheme and the inherent redundancy of the shared (inner) line segments. Therefore, it is necessary to decrease that value by the number of instances minus 2, albeit preserving the end components. The following formula states that the number of instances (cardinality) of the degree value multiplied by the degree value, minus the quantity of the cardinality minus two, indicates the Order to which that constant-valued *n-tuplet* belongs, which is corroborated by Table 8.

$$(|d| * d) – (|d| - 2) \rightarrow \text{Order}N$$

where	*d*	the constant degree value
	|*d*|	the cardinality of degree values.

Illustrated by the following examples:

{3,3,3}	➔	(|3| * 3) – (|3| - 2) = (3*3) – (3-2) + 1 = 9 – 1 = 8	➔ Order *8*
{4,4,4}	➔	(|4| * 4) – (|4| - 2) = (3 * 4) – (3–2) = 12 – 1 = 11	➔ Order *11*
{6,6,6,6}	➔	(|6| * 6) – (|6| - 2) = (4*6) – (4-2) = 24 – 2 = 22	➔ Order *22*

ANALYSIS

Assuming adequate computing resources, the SAS solution can generate trees of Order *N*. Table 9 shows the results of the analyses from Order 6 through 25, listing the number of Candidate and Unique trees, the computing run-time, juxtaposed with known actual trees. Notice that the elimination of redundant tress seems to stay around fifty percent, which makes sense when considering the mirror image of HMI-trees. The number of unique trees increases dramatically as the Order *N* increases, soon consisting of thousands of trees, as well as the computer run time going from seconds to hours for a single analysis.

HMI-Trees Based on SAS Solution

	------------------- Trees	-------------------		Run Time	# Actual	
Order *N*	**# Candidates**	**# Unique**	**% Decrease**	**(hh:mm:ss)**	**Trees**	
6	2	2	0.0	--	2	
7	3	2	33.3	--	2	
8	5	4	20.0	--	4	
9	8	5	37.5	--	5	
10	13	9	30.7	--	10	←
11	21	12	42.8	--	12	
12	34	21	38.2	--	26	←
13	55	30	45.4	--	30	
14	89	51	42.6	--	Unknown	
15	144	76	47.2	0.08	"	
16	233	127	45.4	0.11	"	
17	377	195	48.2	0.16	"	
18	610	322	47.2	0.85	"	
19	987	504	48.9	1.64	"	
20	1,597	826	48.2	15.82	"	
21	2,584	1,309	49.3	37.98	"	
22	4,181	2,135	48.9	6:58.83	"	
23	6,765	3,410	49.5	16:55.44	"	
24	10,946	5,545	49.3	3:03:25.36	"	
25	17,711	8,900	49.7	7:35:22.27	Unknown	

Table 9: Analyses of HMI-Trees from Order 6 to 25.

Figure 1 illustrates the exponential increase of HMI-Trees, plotting the number of trees, as listed in Table 9, along with its log transformation, by Order *N*. Clearly, it would require substantial computing resources to determine a collection of HMI-Trees for Order N above fifty.

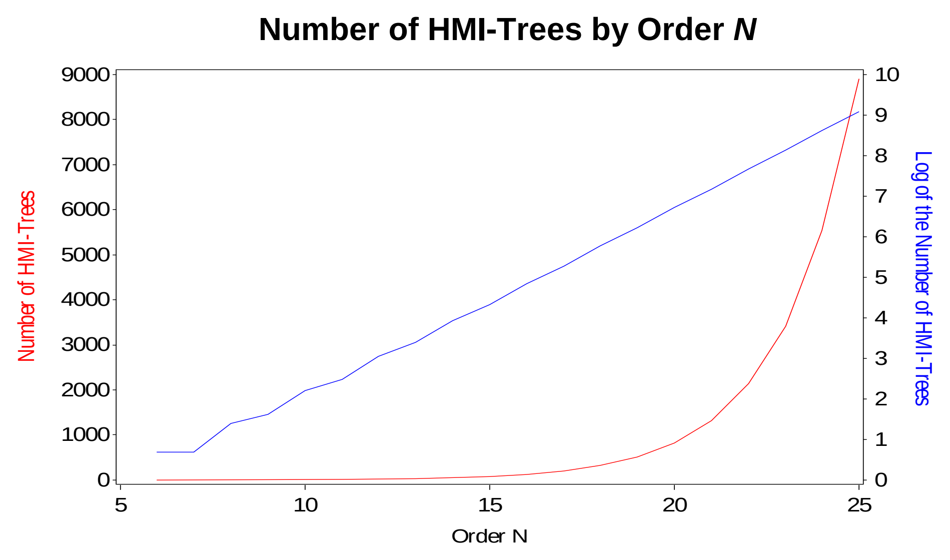


Figure 1: Plot showing the frequency of HMI-Trees by Order, from 6 to 25.

GENERATING WHEN CLAUSES

Assume that you would like to generate the collection of trees for Order 30. How many WHEN clauses would be needed to perform that analysis? Without the necessary WHEN clauses, the program cannot generate all the candidate trees. Table 10 shows some empirical evidence indicating the number of WHEN clauses needed for generating HMI-Trees of Order ranging from 4 to 11. The formula for computing the number of WHEN clauses for Order *N* uses the SAS integer function, as follows:

$$\text{INT}((\&\text{N.} - 2) \ / \ 2) \rightarrow \# \text{ WHEN Clauses}$$

Thus, for example, there must be four WHEN clauses to generate HMIT-Trees for Order 10 or 11. However, the task of generating WHEN clauses requires much more work, such as the inclusion of the logic statement buried inside the nested DO-loops.

Number of WHEN Clauses for Order 4 Through 11

ORDER	ENODES	INODES	TLINES	MLINES	# WHEN Clauses
4	3	1	3	3	1
5	4	1	4	3	1
6	5	1	5	3	2
	4	2	6	6	
7	6	1	6	3	2
	5	2	7	6	
8	7	1	7	3	3
	6	2	8	6	
	5	3	9	9	
9	8	1	8	3	3
	7	2	9	6	
	6	3	10	9	
10	9	1	9	3	4
	8	2	10	6	
	7	3	11	9	
	6	4	12	12	
11	10	1	10	3	4
	9	2	11	6	
	8	3	12	9	
	7	4	13	12	

Table 10: The number of WHEN clauses needed to process HMI-Trees of Order 4 through 11.

In order to generate the WHEN clauses needed, review an explicit WHEN clause, as shown below. Notice the multiple DO statements and their respective END statements, along with the IF statement that tests whether the tree is a viable candidate. The objective is to write a SAS macro that generates all the WHEN clauses accordingly, as needed for Order *N*.

```
when(%eval(&n.-4)) do;
   do inode1 = 3 to enodes;
      do inode2 = 3 to enodes;
         do inode3 = 3 to enodes;
           do inode4 = 3 to enodes;
               if (inode1 + inode2 + inode3 + inode4) eq tlines
                  then output;
               end;
            end;
         end;
      end;
   end;
```

The macro %gen_when performs the task by first assigning the macro variable NWHEN indicating how many are needed for Order *N*, then proceeds to build each WHEN clause in three phases: the DO statements, the IF statement, then the END statements. The macro definition uses the SUM function, which facilitates building the Boolean expression of the IF statement.

```
%macro gen_when;
   %let nwhens = %sysfunc(int((&n.-2)/2));
   %do i = 1 %to &nwhens.;
      when(%eval(&n.-&i.)) do;
         %do j= 1 %to &i.;
            do inode&j. = 3 to enodes;
            %end;
               if sum(of %do k = 1 %to &i.; inode&k. %end;) eq tlines
                  then output;
            %do j = 1 %to &i.;
               end;
               %end;
         end;
      %end;
%mend gen_when;
```

The previous Data step with a lengthy SELECT/WHEN statement needed to accommodate a predetermined Order limit of HMI-Trees has been supplanted with a Data step that contains the %gen_when macro, which generates as many WHEN clauses, as needed, dynamically. Programming note: Using a semicolon after the %gen_when macro causes the program to bomb because it disrupts the syntax requirement of the SELECT/WHEN statement.

```
 data ds01;
   length _numeric_ 3;
   do enodes = %eval(&N.-1) to 1 by -1;
      inodes = &N. - enodes;          * Number of internal nodes   ;
      tlines = inodes + (&N.-2);      * Total lines required       ;
      mlines = 3 * inodes;            * Minimum lines required     ;
      if tlines ge mlines
         then ;
         else leave;                  * Test for Minimum Threshold ;
      select(enodes);

         %gen_when      ← Generates all the necessary WHEN clauses ;

         otherwise;
         end;
      end;
   drop inode: enodes;
run;
```

CONCLUSION

The proposed SAS solution fails to discern more than one HMI-Tree from a single notation, such as the tree {3,3,3,3}, from which two trees can be drawn. Thus, one improvement of the application would be to determine whether a single notation represents more than one tree. Another improvement for analysis would be to actually draw the HMI-Trees, rather than write them in notational format. Finally, a formula for computing the cardinality of Order *N* should be obtained as a validation feature.

Generating HMI-Trees of Order N requires a lot of effort, even for a computer. The proposed SAS solution creates a superset of HMI-Trees many of which are topologically equivalent; consequently these must be discarded. The notational format offers an intuitive approach to writing the trees premised on the idea that a tree is a collection of partitioned components centered on an internal node. Despite the caveat of not being able to discern multiple trees from a single notation, the SAS solution generates an exact minimum of HMI-Trees.

REFERENCES

Herke, Sandra, “Graph Theory: 40. Cayley's Formula and Prufer Seqences part 1/2".   YouTube, https://www.youtube.com/watch?v=Ve447EOW8ww.

"The Problem in Good Will Hunting - Numberphile." YouTube, 04 Mar. 2013. Web. 15 May 2014.

Proof Involving a Problem from “Good Will Hunting”." Graph Theory. N.p., n.d. Web. 15 May 2014.

"Series-Reduced Tree." -- from Wolfram MathWorld. N.p., n.d. Web. 15 May 2014.

CONTACT INFORMATION

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

Name: John R. Gerlach
E-mail: JRGerlach0907@gmail.com

APPENDIX – SOLUTIONS FOR N=11, 12