ABSTRACT

Your data analysis projects can use the SAS® LASR™ Analytic Server and its new HYPERGROUP functionality to mine relationships using graph theory. Discover which data entities are related and, conversely, which sets of values are disjoint. In cases when the sets of values are not completely disjoint, HYPERGROUP can identify data that is strongly connected and identify “neighboring” data that is weakly connected, or data that is a greater distance away. Each record is assigned a hypergroup number and, within hypergroups, is assigned a color, community, or both. The GROUPBY facility, WHERE clauses, or both can act on a hypergroup’s number, color, or community to conduct analytics using data that is “close”, “related”, or more “relevant”. The algorithms that are used to determine hypergroups are based on graph theory. We show how the results of HYPERGROUP allow equivalent graphs to be displayed and useful information to be seen, aiding you in controlling what data is required to perform your analytics.

INTRODUCTION

SAS LASR Analytic Server can handle really big data. It does so by distributing data across a grid, and using analytical algorithms specially designed to harness the computing power of a grid, executing in multiple threads on every grid computer. Sometimes data is so big that determining reasonable ways to reduce its size is necessary to reduce runtimes. Also, breaking apart data is sensible in that analytic tasks use data that’s related in some way. You make better business decisions, faster. Crucial data structure can be unearthed and seen. A case study using Facebook data is included in the last section of the paper.

AN APPLICATION

HYPERGROUP is useful in a wide variety of applications. One that springs to mind is a business that makes recommendations to customers as to what they should buy. A LASR table can have a variable containing customer names, and another variable for some commodity they bought, borrowed, or rented, or an offer to which the customer responded.  There might be other variables that can be used to separate data such as customer traits (for example, location, age, income) or commodity traits such as model or price. By determining hypergroups, you learn how data has disjoint sets of values for variables, enabling analyses to be conducted per hypergroup.

Imagine a LASR table that has a column for customer names, and another column for vehicles they bought. By hypergrouping these two variables, an output LASR table is produced that is usually similar to original data with the addition of a column called _HypGrp_ that has values 0, 1, 2, ... so that records that belong to the same hypergroup have the same _HypGrp_ value. You can, for example, see the records of vehicle buyers who all bought one or more vehicles belonging to the same set of vehicles. If data has two hypergroups, then there are no records that indicate any of hypergroup 0's customers bought any hypergroup 1 vehicles, and no hypergroup 1 customers bought any hypergroup 0 vehicles.

You do not supply the sets of values. There might be no way you could. Disjoint sets of values are found by HYPERGROUP. In the example, you do not know beforehand whether some customer is a hypergroup 0 customer or a hypergroup 1 customer, nor beforehand whether some vehicle is a hypergroup 0 vehicle or a hypergroup 1 vehicle.

Once HYPERGROUP has been run, imagine a new customer arrives on the scene; you can determine whether he is more like existing customers of either hypergroup 0 or 1 and market vehicles we presume he would prefer to buy belonging to hypergroup 0 or 1 vehicles, respectively. Alternatively, if you
manufacture a new vehicle that is similar to either hypergroup 0 or 1 vehicles, you might best expend marketing effort to customers in hypergroup 0 or 1, respectively.

Hypergroup technology is often used in “Recommender” systems. When you are shopping at an Internet retailer, they provide you with items they recommend you also consider buying. That’s based on your previous buying history and, as important, customers like you.

**ANOTHER APPLICATION**

In customer/campaign/account/insurance policy management applications, and other data cleansing/standardization of data applications, data may have a column for customer names, and another column for address. If these columns are hypergrouped then, for example, records that have these values in the name column: John Smith, J. Smith, John A. Smith, John Smith Jr., Dr. Smith, The Right Honorable John Archibald Smith, Esq., and have the same value, 123 Main St., for the address column, is determined by the Hypergroup action to belong to the same hypergroup.

Now imagine this individual (who goes by several names in your data) owns a vacation home at 456 Beach Rd. This becomes known because of a record in the data that has one of the names he goes by and the value 456 Beach Rd. in the address column, although you’d better check this is not another man named John Smith. Suppose it’s the same guy. Now all records that have any one of the John Smith name variations in the Name Column, and either of the two addresses in the Address column are determined to belong to the same hypergroup.

If that is the extent of the hypergroup, then there are no records that indicate this individual lives anywhere other than these two addresses. There are no records that indicate any other person lives at either of these two addresses.

Now imagine the table has a third column whose values are telephone numbers. Hypergroup all three columns: name, address, and phone number, and interesting associations may be revealed. For instance, if there is a record in data that has address 456 Beach Rd. and phone number 555-1212, then you can imply that phone number belongs to John Smith. If there is a record that has the value J. Smith PhD in the name column and 555-1212 in the phone number column, then that’s another name he has used, and whenever a new association is discovered, that in turn may cause yet more associations to be discovered.

If there is a record with name Harry Smith and phone number 555-1212, then this could reveal a valid connection, or not, if data was incorrectly entered. In fraud detection applications, such connections are of interest, possibly suspicious. (The Right Honorable Dr. John Archibald Smith, Esq. is using the pseudonym Harry Smith to commit a fraud- surely not?)

Similar applications have been used to detect and investigate criminal and terrorist networks. For managing promotional campaigns, hypergrouping allows you to send offers to only one member of a household.

**THE HYPERGROUP ACTION**

The SAS program below puts PROC IMSTAT to work determining hypergroups. In the following data set, variables varA to varD are the hypergroup variables, and the remaining variables are other variables you need to appear in an output LASR table.

The varA to varD variable values have values beginning with A and Z, B and Y, C and X, and so on. In our example that distinguishes different hypergroups. However, in real-world data, hypergroups can rarely have such obvious distinctiveness, and can rarely be identified by eye, nor can data for hypergroups reside in consecutive records, on the same LASR server nodes. Hypergroups cannot usually be identified and highlight-colored as in the following DATA step. It’s rare that hypergroups can be indicated by WHERE clauses or as ordinary GROUPBY variables without hypergroups first being determined.

```sas
data arc1;
  input varA $ varB $ varC $ varD $ extra1 extra2 $ extra3 $ extra4 @@;
```
data; 
A00    A01    A04    Z05    0 A Z    5 0 C 6    X09 4 A Z    9
A00    A02    A06    0 A Z    6 0 C 6    X09 4 A Z    9
A01    A02    A04    Z07    1 A Z    7 0 C 4    X09 4 A Z    9
A01    A02    A04    Z07    1 A Z    7 0 C 4    X09 4 A Z    9
A02    A03    Z08    2 A Z    8 0 C 4    X09 4 A Z    9
A02    A03    Z08    2 A Z    8 0 C 4    X09 4 A Z    9
A03    Z05    Z06    3 A Z    9 0 C 4    X09 4 A Z    9
A04    Z05    Z07    Z09    4 A Z    9 0 C 4    X09 4 A Z    9
D00    D01    D04    W06    0 D W    6 0 F 6    F00 0 F04 0 F 6 0 F 6
D00    D01    D04    W07    0 D W    7 0 F 6    F00 0 F04 0 F 6 0 F 6
D01    D02    W05    W07    1 D W    7 0 F 6    F00 0 F04 0 F 6 0 F 6
D01    D04    W05    W08    1 D W    8 0 F 6    F00 0 F04 0 F 6 0 F 6
D02    D03    W06    W07    2 D W    7 0 F 6    F00 0 F04 0 F 6 0 F 6
D02    D04    W06    W09    2 D W    9 0 F 6    F00 0 F04 0 F 6 0 F 6
D02    W05    W07    W08    2 D W    8 0 F 6    F00 0 F04 0 F 6 0 F 6
D02    W05    W07    W08    2 D W    8 0 F 6    F00 0 F04 0 F 6 0 F 6
D03    W05    W07    W09    3 D W    9 0 F 6    F00 0 F04 0 F 6 0 F 6
D04    W05    W08    W09    4 D W    9 0 F 6    F00 0 F04 0 F 6 0 F 6

; proc lasr
create force port= &pport data=arc1 path="/tmp/"
performance host= &hhost nodes=3 threads=2; /* you change */
run;
libname mylasr sasiola host=&hhost port= &pport tag=work;

proc imstat data=mylasr.arc1;
hypergroup VarA varB varC varD / vars=(extra4 extra3 extra2 extra1)
save=hyptable;
run;

OUTPUT TEMPTABLES
The HYPERGROUP statement creates at least three temporary LASR tables that are assigned to SAS macros _TEMPLAST_ , _TEMPHYGRP_ , and _TEMPEDGES_.

The _TEMPLAST_ Temptable
In this example, the HYPERGROUP statement was specified as this:
hypergroup VarA varB varC varD / vars=(extra4 extra3 extra2 extra1);

Therefore, the _tempLast_table has these columns:
- a _HypGrp_column with values that are hypergroup numbers (0, 1, 2, ...).
- the variables VarA varB varC varD specified between hypergroup and the slash, or if there is no slash, the semicolon.
- the VAR=(...) columns--if there are any--extra4 extra3 extra2 extra1. These are columns that are not hypergroup columns that you want to appear in output tables.

table mylasr.&_templast_
fetch / from=1 to=150;
run;

Selected Records from Table _T_8651BD0F_7F9390521138
_HypGrp_  varA  varB  varC  varD extra4 extra3 extra2 extra1
  0 A00  A01  A04    Z05  5.000000  Z A  0
  0 A00  A02  A04    Z06  6.000000  Z A  0
  0 A01  A02  A04    Z07  7.000000  Z A  1.000000
  0 A01  A04  Z05  Z07  7.000000  Z A  1.000000
  0 A02  A03  Z05  Z08  8.000000  Z A  2.000000
The _TEMPHYPGRP_ Temptable

The _tempHypGrp_ temp table has records pertaining to values of the hypergroup variables. It has at a minimum these columns:

- _Value_ : values of hypergroup variables. These are graph vertex names.
• _Index_: vertex indices (0, 1, 2, …).
• a_HypGrp_: hypergroup numbers (0, 1, 2, …).
• _indexH_: vertex indices (0, 1, 2, …) but only within hypergroup subgraphs.
• _XCoord_ and _YCoord_: the coordinates of vertices.

```
table mylasr.&_temphypgrp_;  
    fetch / from=1 to=60;  
run;
```
The _TEMPEDGES_ Temptable

Consider the first record in data:

<table>
<thead>
<tr>
<th>varA</th>
<th>varB</th>
<th>varC</th>
<th>varD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A00</td>
<td>A01</td>
<td>A04</td>
<td>Z05</td>
</tr>
</tbody>
</table>

This record generates three edges: (A00, A01), (A01, A04), and (A04, Z05).

The _TempEdges_ table has all the variables in _tempLast_, except instead of having the hypergroup variables (in this example, varA, varB, varC, and varD), there are variables: _Source_ and _Target_ that have values associated with the origin and destination of edges, and index and coordinates variables associated with the source and target vertices.

The _tempEdges_ temp table has at a minimum these columns:

- a _HypGrp_: hypergroup numbers (0, 1, 2, ...).
- _Source_ and _Target_: the names of the vertices the edge connects.
- _Sindex_ and _TIndex_: vertex indices (0, 1, 2, ...) of _Source_ and _Target_.
- _SindexH_ and _TIndexH_: vertex indices (0, 1, 2, ...) of _Source_ and _Target_ but only within hypergroup subgraphs.
- _XCoordS_ and _YCoordS_: the coordinates of the _Source_ vertex.
- _XCoordT_ and _YCoordT_: the coordinates of the _Target_ vertex.

Selected Records from Table _T_IE114D6E_7FC4D3FB21A0

<table>
<thead>
<tr>
<th><em>HypGrp</em></th>
<th><em>Source</em></th>
<th><em>Target</em></th>
<th><em>Sindex</em></th>
<th><em>TIndex</em></th>
<th><em>SindexH</em></th>
<th><em>TIndexH</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A00</td>
<td>A01</td>
<td>0</td>
<td>1.000000</td>
<td>0</td>
<td>1.000000</td>
</tr>
<tr>
<td>0</td>
<td>A01</td>
<td>A04</td>
<td>1.000000</td>
<td>4.000000</td>
<td>1.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>0</td>
<td>A04</td>
<td>205</td>
<td>4.000000</td>
<td>55.000000</td>
<td>2.000000</td>
<td>3.000000</td>
</tr>
<tr>
<td>0</td>
<td>A00</td>
<td>A02</td>
<td>0</td>
<td>2.000000</td>
<td>0</td>
<td>9.000000</td>
</tr>
<tr>
<td>0</td>
<td>A02</td>
<td>A04</td>
<td>2.000000</td>
<td>4.000000</td>
<td>9.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>0</td>
<td>A04</td>
<td>206</td>
<td>4.000000</td>
<td>56.000000</td>
<td>2.000000</td>
<td>7.000000</td>
</tr>
<tr>
<td>0</td>
<td>A01</td>
<td>A02</td>
<td>1.000000</td>
<td>2.000000</td>
<td>1.000000</td>
<td>9.000000</td>
</tr>
<tr>
<td>0</td>
<td>A02</td>
<td>A04</td>
<td>2.000000</td>
<td>4.000000</td>
<td>9.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>0</td>
<td>A04</td>
<td>207</td>
<td>4.000000</td>
<td>57.000000</td>
<td>2.000000</td>
<td>8.000000</td>
</tr>
<tr>
<td>0</td>
<td>A01</td>
<td>A04</td>
<td>1.000000</td>
<td>4.000000</td>
<td>1.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>0</td>
<td>A04</td>
<td>205</td>
<td>4.000000</td>
<td>55.000000</td>
<td>2.000000</td>
<td>3.000000</td>
</tr>
</tbody>
</table>
STRONG AND WEAK CONNECTIONS

Big data is messy. Consider the data regarding customers who bought vehicles. Suppose some customers (set C₀) usually bought some vehicles (set V₀), and other customers (set C₁) usually bought another set of vehicles (set V₁). These are strong links. However, there might be weak links, related to data that indicates C₀ customers who occasionally bought V₁ vehicles, and C₁ customers who occasionally bought V₀ vehicles. These purchases are outnumbered by the strong link purchases. If it were not for the weak links, C₀ and V₀ would be one hypergroup, and C₁ and V₁ another hypergroup, but as there are, all data might be one great big hypergroup.

You do not supply the sets of values. There might be no way you could. In this example, HYPERGROUP determines the C and V sets by two different algorithms, and outputs color number or community number 0 for records for C₀ V₀, and outputs a color number or community number 1 for records for C₁ V₁. You do not know beforehand whether some customer is a C₀ or C₁ customer, nor beforehand whether some vehicle is a V₀ or V₁ vehicle.

The algorithms used to determine hypergroups, and strong and weak links (and therefore colors and communities) within each hypergroup, are based on graph theory. Vertices have names that are values of hypergroup variables, and edges that connect vertices (value_a, value_b) are generated if there is any record in data that has value_a and value_b as values in adjacent hypergroup list variables.

- finds whether data can be broken into separate parts— that is, separate hypergroups. In graph theory terms, HYPERGROUP determines whether the graph has disconnected components.
- within a hypergroup, or the entire graph if there are no separate hypergroups, HYPERGROUP determines coordinates of the vertices so that the graph can be displayed, and can be used to zoom.
into a particular part of the graph, consequently be a part of data, and subsequently perform analytics.

- determines colors, communities, or both, which can be used to color vertices and edges when graphs are displayed, or to zoom into a particular part of the graph and consequently a part of data, and subsequently perform analytics.

- produces structural graphs, as well as coordinates for them, and output details about strong subgraphs (for example, number of vertices strongly connected, and the number of actual edges that make those connections) and weak links (for example, number of actual edges that weakly connect strong subgraphs).

- computes various centrality measures, often used within a social network analysis application, and for other practitioners who wish to further refine what data is to be used by subsequent analytics.

There are many options you can specify in the HYPERGROUP statement to control how layouts, colors, communities, structural graphs, and centrality are performed. The output temp tables will have more columns in them (for example, you’ve seen the _XCoord_ and _YCoord_ column in _TEMPHYPGRP_). There might also be _Color_, and _Community_, and more. You can also choose to skip some of these tasks that are ordinarily done.

Structural graphs and centrality are covered later in this paper.

Here are more graphs that have circular, mesh, clover leaf: and later you’ll see tree structures:
HYPERGROUP allows you considerable control over the way vertex or edge separators are determined, and the resulting graph partitions:
REMOVING NOISE AND CONFUSION

Consider the following graph whose vertices and colors have been determined with HYPERGROUP's default algorithms:

While none of the vertices are laid out on top of each other, which is good, some vertices with the same color are not very close, which is bad. Has HYPERGROUP failed to determine appropriate vertex color numbers? Is this really a graph that has no exploitable structure? You might think so, given the way it looks.

When the graph is displayed so that only vertices with the same color are prominently displayed, we see many subgraphs like these:
In this example, if you were to draw a line from the top left to the bottom right, the vertices that have the same color are all on one side of that line or the other, and there are edges to vertices with different color that seem to lie on or near that line. These are in the same set of vertices. Only a small number of clicks in the network displayer are required to identify the vertices that have value zero in the _Color_ column, and those that form a separator:

When separators are determined, HYPERGROUP strives to keep the number of vertices in the separator low. However, these vertices can have high degree, and often that is desirable. By the way, the degree of a vertex is the number of edges that have that vertex as one of the vertices the edge connects.
Rerun HYPERGROUP when the separator and edges that are incident onto the separator’s vertices are removed; this is very easy to program by using a simple WHERE statement. The result is that HYPERGROUP determines the graph is now many disconnected hypergroup graphs, such as these:

Imagine you are in charge of an Internet retailer. Whenever a customer shows interest in some item of merchandise, other products are shown to him to tempt him to buy one or more products. These recommended products were bought by other customers who have similar buying behavior to the prospective purchaser. However, the least useful data is for customers who seem to have no particular buying behavior; they buy lots of products that were selected from a wide range of product categories. Much better recommendations can be made when data for these “noisy” customers is removed.

Now imagine you are in charge of an advertising budget and must decide on which websites you should advertise. If the vertices in the above graph represent people and websites, then advertising on separator websites could yield the best results.

Obviously, separators separate data into two or more pieces. There is a wide range of applications when breaking apart data in a meaningful way is desired (for example, to improve reliability in communications systems by identifying places where failure would be serious, or to conduct military operations with more efficient interdiction, or simply because the data is too big to be tackled all at once).

**WHEN DATA IS BIGGER**

Consider data that has been associated with this graph that has one-quarter million edges, 25 thousand vertices, and that has a tree-like structure where every subgraph is weakly connected to three descendant subgraphs and, of course, one parent. The figure on the right shows only the weak edges:
It might be more important to know how strongly connected subgraphs are connected to other strongly connected subgraphs by weak edges, than how strong edges connect vertices within the strong subgraph. Indeed, the latter might be unimportant. Therefore, you can indicate to HYPERGROUP that structural graphs are to be produced. A graph is formed that consists of a vertex for each color (or community)—that is, for each strong subgraph, and one and only one edge for between \( c_i \) and \( c_j \) if there exists one or more edges in the real graph between any vertex with color (or community) \( c_i \) and any vertex with color (or community) \( c_j \). You can do the following:
• use HYPERGROUP to determine hypergroups and colors (or communities), but not coordinates, and create the structural=color (or community) graph, and determine the layout of it, and then

• if a particular hypergroup-color (or community) subgraph is of particular interest, use HYPERGROUP again for that hypergroup-color (or community) fell swoop.

This strategy might save time in that only what you need is determined. Also, in effect, only relevant subgraphs are attacked by HYPERGROUP, and even when there are several to be attacked, the overall time might be much less than doing everything all in one "fell swoop". You can drill down into parts of the data that are of more interest than other data.

It's now clear that the structure of the graph is a tree in which vertices have three descendants.

Some of the vertices have been labeled with their colors, and you can easily rerun HYPERGROUP, or for that matter any action, to focus attention onto one or a subset of records that have a particular group-by value combination, hypergroup number, and color values.
FOCUSING UPON DATA FOR PARTICULAR HYPERGROUPS

To identify which hypergroup is associated with a hypergroup variable value, use the WHERE, FETCH, and STORE statements. For example, if analyses are to focus on the hypergroup in which a hypergroup variable has value "C04", do this:

```plaintext
table mylasr.&_temphypgrp_;  
  where _Value_="C04";  
  fetch / save=fetchTab from=1 to=60;  
  store fetchTab(1,cols=_HypGrp_)=hgmacro /  
    left="(" control="_HypGrp_=%" separator=" or " right=")";  
run;  
%put &hgmacro;
```

In this example, the hgmacro expands to \((\_HypGrp_=2)\)

Now hgmacro can be used in a WHERE statement that governs how _TEMPLAST_ is read to yield output for only the hypergroup with "C04".

```plaintext
table mylasr.&_templast_;  /* or many other tables created by Hypergroup */  
  where &hgmacro ;  
  fetch / from=1 to=150;  
run;
```

Notice that some records do not have the C04 value anywhere in them, yet they belong to the same hypergroup as those that do.

The same techniques can be used to obtain data for even greater granularity; for a particular value combinations of Group-By/Hypergroup/Color, or Group-By/Hypergroup/Community, or Group-By/Hypergroup/vertices that have coordinates in a certain region, or Group-By/Hypergroup/being close by virtue of centrality measures.

We describe the topic of centrality in the next section.

CENTRALITY

The centrality of a vertex quantifies the importance of that vertex among its peers. Centrality can also be considered a measure of the seriousness of removing a vertex (and edges incident onto it).

There are many ways centrality can be determined, and it's often the case that one centrality more than another is better suited to some graphs, but for another graph, another centrality is more useful than the first.

Of the dozens of centralities, HYPERGROUP determines four centralities in common use based on shortest paths (Graph Centrality, Stress Centrality, Closeness Centrality, Betweenness Centrality), and another centrality (Centroid Centrality) based on the layout of a graph.

The shortest path between two vertices is the smallest number of edges in a path from one vertex to the other.
THREE DIMENSIONS

Up until now, we have assumed that only two dimensions are required. If you need to work in three dimensions, simply specify the HYPERGROUP statement's THREED option (I tried to name this option 3D but SAS won’t allow it). When in 3-D, whenever there are x and y coordinates, there will now be an additional z coordinate. Color, community, and the shortest path-based centrality computations remain much the same as in the 2-D case, but obviously the layout algorithms are different, as are the centroid centrality results. Graphs are often seen to be more compact as space between vertices is obtained by using the extra dimension.

At the moment, SAS software products do not have a 3-D graph renderer, but visually appealing and informative details of 3-D graphs can be realized by the following:
- continuing to use 2-D graph renderers that are available in SAS such as SAS/GRAPH® NV Workshop, which has been used for all 2-D graphs so far in this paper,
- using 3-D scatter plotters to display vertices: I've opted to use the JMP® 3-D plotter that has the feature that 3-D scatter plots can be tilted and rotated interactively, or
- running a JMP script that plots vertices onto a 3-D scatter plot, and then adds vectors for every edge you want displayed.

Consider the graph from the last section (here shown are centroid centrality's angles with respect to the graph's centroid):

Looking down from the positive z-axis (using the JMP 3-D scatter plotter) (left), and using SAS/GRAPH NV Workshop, using only the _Xcoord_ and _Ycoord_ columns (right).
Looking down from the positive y-axis (using the JMP 3-D scatter plotter) (left), and using SAS/GRAPH NV Workshop, using only the _Xcoord_ and _Zcoord_ columns (right).

To save space, we’ve omitted the illustration that looks down from the positive x-axis.

Here are screen shots when axes are tilted and rotated:
For the next few figures, the coordinates HYPERGROUP uses are the JMP 3-D scatter plotter to plot vertices. The communities HYPERGROUP determines are used by the scatter plotter to set the colors of the vertices.

Another example, a 4x4x4 mesh: There are 64 strong regions, each having 50 vertices connected by 1000 edges. There are 144 sets of weak edges, each having 5 edges.
In previous sections, this graph was used. It has 23 thousand vertices and about one-quarter million edges. There are 460 strong subgraphs that on average have 50 vertices and 533 edges. These strong subgraphs are connected by 459 sets of weak edges that have on average 10 edges.

The structure is that of a tree with vertices having three descendants. Here are screen shots of it in 3-D:
SEEING EDGES IN 3-D GRAPHS

A JMP script must be written (contact me if you want it) so that graph edges can be seen. Here are screen shots of a small 3-D graph with all edges shown. Note that strong edges are colored the same as the vertices they connect. Weak edges are black.

It’s a pity this doc is not like pictures in books and newspapers, and portraits on walls in Harry Potter movies that move, because then you’d see the screenshots tilt and rotate, and you’d clearly see the 3-D graph with squadrons of vertices flying together in formation and performing amazing aerobatics, as if on broomsticks playing Quidditch!
Even for graphs that are quite small (and just as it is the case with 2-D), larger graphs become indistinct when too many edges are shown. It's often better to show all vertices, but only weak edges:
If this is still too cluttered, HYPERGROUP can, as an option, create structural graphs based on partition, community, or both. One vertex represents an entire partition or community, and an edge represents all weak edges between two of them. Coordinates are also determined in 3-D.
A CASE STUDY USING FACEBOOK DATA

DATA

In this session, we are going to present a case study using data from Facebook to show how to apply HYPERGROUP to real data. Bear in mind that the HYPERGROUP action can do much more than what is demonstrated in the case study. HYPERGROUP can be used in many applications other than social networking.

The data set is Facebook “friends” data from Stanford Network Analysis Project. (http://snap.stanford.edu/data/egonets-Facebook.html). All of the user information is anonymized. There are 4039 nodes in the data set, of which each node is a Facebook account. They are indexed from 0 to 4038. There are 88,234 edges, which means if two Facebook accounts “friend” each other, there is an edge between them. So obviously, it is very dense network data, which is not surprising considering how people use Facebook (often, a user has a large circle of friends). There are only two columns in the data set, _from_ and _to_, which are the index of the source account of the edge and the index of the target account of the edge. For example, if a row value of _from_ is 178, and value of _to_ is 303, then that means there is an edge between the account who indexed 178 and 303. In other words, account index 178 and user index 303 are “friends”.

DETAILS

With this “friendship” network data, we are going to show how to do the following tasks in just one HYPERGROUP statement.

- Determine the layout of the social network for visualization.
- Conduct graph partitioning that partitions the data into several subgraphs so that the users in each subgraph are strongly connected, though they might not directly “friend” each other. The subgraph number is indexed as '_color_' in the result table.
- Conduct community detection. It is similar to graph partitioning in the above point, but uses a completely different algorithm. The community is indexed as '_community_' in the result table.
- Measure the centralities of the user and identify the most influential users.

Assuming you have already started a LASR server and loaded the data into it, you can use the following PROC IMSTAT code to run HYPERGROUP in SAS® In-Memory Statistics.

```sas
proc imstat;
  table mylasr.facebook_edges;
    hypergroup (_FROM_ _TO_) / structural=both centrality
      layout=walshaw niter=100 scalecoords nopendants
      width=101 length=99 margin=2
      graphpartition highdegree=1 maxnodes=150 separator=edges
      commiter=25 commalg=LLsemisynchronous;
run;
  table mylasr.&_temphypgrp_
    fetch / orderby=(_Betweenness_ ) descending=_Betweenness_
    distinct _HypGrp_ _Color_ _Community_
    promote temphypgrp;
run;
quit;
```

In line [1], the IMSTAT procedure is invoked. It is an interactive procedure that keeps running until you quit it by issuing the QUIT statement or starting other procedures or DATA step sessions. After you terminate PROC IMSTAT, the data is not removed from the LASR server memory. You can start a new IMSTAT procedure to continue your analytics without reloading data. However, temporary tables will be removed when the IMSTAT procedure ends.
In line [2], the TABLE statement indicates to the SAS LASR Analytic Server which LASR table you want to work on. Before the next TABLE statement is claimed, all actions will run on this table. Therefore, you can run several actions on the same LASR table, without specifying the LASR table again and again.

In line [3], the HYPERGROUP statement uses two input variables in the parentheses, _from_ and _to_. The STRUCTURAL option specifies which structural analysis you want to conduct, graph partition, or community, or both, or even none. In this case, both graph partition and community detection are conducted. The CENTRALITY option tells HYPERGROUP that you want to calculate centrality measures of vertices.

In line [4], the options determine the layout algorithms and parameters of the algorithm.

In line [5], the options determine the width, length, and margin of the layout.

In line [6], the commonly used parameters of the graph partition algorithm are specified.

In line [7], the algorithm and the number of the iteration of community detection are specified.

Please note that all the options after the slash ‘/’ in a HYPERGROUP statement are irrelevant to the order they are specified.

In line [8], the TABLE statement tells SAS LASR Analytic Server that we want to work on the temporary table _temphypgrp_ produced by HYPERGROUP.

In line [9], the top 20 vertices with the largest values of betweenness centrality measure are printed in descending order.

In line [10], distinct counts of _HypGrp_, _Color_, and _Community_ are calculated.

In line [11], the temporary table is promoted as a ‘permanent’ LASR table in memory.

**OUTPUT TABLES**

The following table shows the first 20 rows in the _TEMPHYPRP_ temporary table. The vertices are ordered by betweenness centrality so that these are the top 20 most influential Facebook accounts by betweenness centrality. You can also find hypergroup numbers in column _HypGrp_, X and Y coordinates in column _XCoord_ and _YCoord_, subgraph number in _Color_, community number in _Community_, and the centrality measures.
Let us take a closer look at the result by calculating the distinct counts of _HypGrp_, _Color_, and _Community_ columns. The following table shows there is only one hypergroup, which means all of the Facebook accounts are somewhat connected. In other words, any two accounts are linked together through some path or paths. Within the hypergroup, there are 219 subgroups and 49 communities.

<table>
<thead>
<tr>
<th>Column</th>
<th>Number of Distinct Values</th>
<th>Number of Missing Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>HypGrp</em></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><em>Color</em></td>
<td>219</td>
<td>0</td>
</tr>
<tr>
<td><em>Community</em></td>
<td>49</td>
<td>0</td>
</tr>
</tbody>
</table>

PLOTS

There are many ways to render results of HYPERGROUP. They can be rendered in the following:

- SAS® Visual Analytics.
- SAS/GRAPH NV Workshop.
- Some SAS procedures such as SGPLOT, G3D, SAS JMP and so on.
- Any other tools that can render coordinates, colors, or edges.

The following two plots are rendered by scatter plot nodes in SAS Visual Analytics. The vertices are nicely distributed across the diagram, with colors specified by _color_, which is subgraph number, and _community_, respectively.
Community Detection

The following two plots are rendered by scatter plot nodes in SAS/GRAPH NV Workshop. Because this software product is dedicated to displaying networks, it can render edges, which scatter plotting procedures cannot, except that JMP’s scatter plot can be called by a JMP script to superimpose edges. Hence, it would be easy to visualize the accounts that play as critical roles.

For example, some vertices have many connections within their subgraph and community. Some vertices not only have many connections, but also they are very critical in the way that many paths go through them so that they link two big clusters of vertices together. In the example, colors of edges are determined by color of source vertices. It is very obvious from the following two results that the graph partition algorithm partitions the graph into many more subgraphs than the community detection algorithm.
The following plot is rendered by the G3D procedure where each color represents a community. In a three-dimensional plot, the vertices can spread out to reduce overlap of communities in the visualization so that the visualization is much clearer.

CONCLUSION

The algorithms used to determine hypergroups are based on graph theory. HYPERGROUP functionality generates one or more graphs based on your data, and produces results so that the graphs can be displayed and useful information can be derived. Data structure can be unearthed. You can control what data is supplied to analytics.

HYPERGROUP determines graph coordinates, colors, communities, structural graphs, centrality measures, whether for two-dimensional or three-dimensional graphs.

HYPERGROUP functionality for SAS LASR Analytic Server uses this powerful new tool to discover how data can be classified or assigned, so that records have columns that have disjoint sets of values. In cases when the sets of values are not completely disjoint, HYPERGROUP can identify data that is strongly connected, and identify “neighboring” data is weakly connected, or data that is farther away than that.

Each record is assigned a hypergroup number, and within hypergroups a color, community, or both. The GROUPBY facility, WHERE clauses, or both can act on hypergroup number, color, or community in order to conduct analytics using data that is “related” or more “relevant”. Your analysis is better, results can be obtained faster, and you really do have SAS®... THE POWER TO KNOW®.
REFERENCES

ACKNOWLEDGMENTS
We would like to express our gratitude to Juthika Khargharia for providing the PROC G3D code to render the three-dimensional plot of HYPERGROUP results.

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

Trevor Kearney
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Trevor.Kearney@sas.com
http://www.sas.com

Yue Qi
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Yue.Qi@sas.com
http://www.sas.com

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.