SAS® OPTGRAPH
Procedure 14.1
High-Performance Features
Contents

Chapter 1. High-Performance Features of the OPTGRAPH Procedure . . . . . . . . . . 3

Index 27
Credits

Documentation

Writing Matthew Galati, Yi Liao, Michelle Opp
Editing Anne Baxter, Ed Huddleston
Documentation Support Tim Arnold, Melanie Gratton
Technical Review Manoj Chari, Charles B. Kelly, Bengt Pederson, Daniel Underwood

Software

PROC OPTGRAPH Matthew Galati, Yi Liao

Support Groups

Software Testing Charles B. Kelly, Daniel Underwood
Technical Support Tonya Chapman
Chapter 1
High-Performance Features of the OPTGRAPH Procedure

Contents

Overview ................................................................. 3
Recommended Workflow .............................................. 4
Single-Machine Mode .................................................... 4
Alongside-the-Database Distributed Mode ......................... 4
Graph Size Limitations ................................................. 5
Controlling the Execution Environment ......................... 5
PERFORMANCE Statement ........................................ 6
Distributing Input Data to the Appliance .................... 7
Community Detection ................................................. 8
COMMUNITY Statement ............................................. 8
Community Detection Details .................................... 10
Example: Community Detection on a Simple Undirected Graph ...... 13
Centrality Computation by Cluster ............................. 17
Example: Centrality by Community for a Simple Undirected Graph ..... 18
Example: Centrality by Cluster for a Simple Undirected Graph ... 19
References ............................................................... 25

Overview

The OPTGRAPH procedure in SAS High-Performance Network Algorithms enables you to perform community detection and centrality computations on large graphs in a high-performance environment. It uses an appliance that houses a massively parallel database management system (Teradata or EMC Greenplum) to manage data in distributed form and to perform computations in parallel on an x64 Linux platform. A computing appliance is a dedicated hardware and software environment that acts as a server to provide computing resources in a client/server model. You connect indirectly to the appliance through the network connection between the client machine and the appliance. Software instructions on the client machine are translated into commands that are run on the appliance. For information about installing SAS High-Performance Network Algorithms, see the SAS High-Performance Analytics Infrastructure: Installation and Configuration Guide. PROC OPTGRAPH runs in either single-machine mode or distributed mode. NOTE: Distributed mode requires SAS High-Performance OPTGRAPH.
Chapter 1: High-Performance Features of the OPTGRAPH Procedure

Recommended Workflow

For graphs that contain hundreds of millions or billions of links, such as a typical telecommunications network, minimizing the movement of data is crucial to achieving maximum performance in a high-performance distributed computing environment. Therefore, the following workflow is recommended when you use PROC OPTGRAPH to perform community detection and to compute centrality metrics on a high-performance appliance:

1. Distribute the links data set to the appliance as described in the section “Distributing Input Data to the Appliance” on page 7. The links data set should be distributed by the variable that represents the from node of each link. Otherwise PROC OPTGRAPH will automatically redistribute the links by from node before the actual computation, which could be time-consuming if the graph contains a large number of links.

2. Run PROC OPTGRAPH with the COMMUNITY statement to perform community detection as described in the section “Community Detection” on page 8. Write the resulting OUT_INTRA_COMM_LINKS= output data set to the appliance. Repeat this step as many times as desired by using different options in the COMMUNITY statement to control the parallel community detection algorithm. For example, you might want to try different values for the maximum community size or the maximum number of iterations. For information about the options that are available when you run community detection, see the section “Community Detection” on page 8.

3. After running community detection, run PROC OPTGRAPH again with the CENTRALITY statement to compute centrality metrics by cluster as described in the section “Centrality Computation by Cluster” on page 17. As the links input data set, use the OUT_INTRA_COMM_LINKS= output data set that was created in step 2.

Single-Machine Mode

Single-machine mode is a computing mode in which multiple processors or multiple cores are controlled by a single operating system and can access shared resources, such as disks and memory. More simply, single-machine mode for high-performance procedures means multithreading on the client machine. In single-machine mode, the OPTGRAPH procedure runs multiple concurrent threads on a multicore machine in order to take advantage of parallel execution on multiple processing units. For information about running the OPTGRAPH procedure in single-machine mode, see the SAS OPTGRAPH Procedure: Graph Algorithms and Network Analysis.

Alongside-the-Database Distributed Mode

Distributed mode is a computing mode in which several nodes in a distributed computing environment participate in the computations. In distributed mode, the OPTGRAPH procedure performs analytics on the database management system (DBMS) appliance. The OPTGRAPH procedure in SAS High-Performance Network Algorithms supports the alongside-the-database model of distributed execution, in which the data are stored in the distributed database and read in parallel from the DBMS.
When the input data are stored in the DBMS and the grid host is the appliance that houses the data, the OPTGRAPH procedure creates a distributed computing environment in which the analytic process is co-located with the nodes of the DBMS. PROC OPTGRAPH then passes data from the DBMS to the analytic process on each node. Instead of moving the data across the network and possibly back to the client machine, PROC OPTGRAPH passes the data locally between the processes on each node of the appliance.

Because the analytic processes on the appliance are separate from the database processes, the technique is referred to as alongside-the-database execution, in contrast to in-database execution, where the analytic code executes within the database process.

Before you run PROC OPTGRAPH alongside the database, you should distribute the data to the appliance. This step is described in the section “Distributing Input Data to the Appliance” on page 7. In the alongside-the-database model, the number of compute nodes is determined by the layout of the database and cannot be modified. Therefore, if you specify a NODES= option in the PERFORMANCE statement in distributed mode, PROC OPTGRAPH ignores it. (Some SAS high-performance procedures support a NODES= option in the PERFORMANCE statement to control the number of compute nodes used; this option is valid only when the procedure passes data from the client to the appliance.)

Graph Size Limitations

PROC OPTGRAPH can handle graphs that contain up to 2,147,483,647 nodes. The maximum number of links it can handle depends on the execution mode in which it runs. In single-machine mode, the maximum number of links is 2,147,483,647. In alongside-the-database distributed mode, the maximum number of links on each distributed node is 2,147,483,647. However, there is no limit on the total number of links across all nodes of the appliance.

Controlling the Execution Environment

You control the execution mode by using environment variables or by specifying options in the PERFORMANCE statement. The important environment variables follow:

- **grid host** identifies the domain name system (DNS) or IP address of the appliance node to which the OPTGRAPH procedure connects to run in distributed mode.

- **installation location** identifies the directory where the SAS high-performance software is installed on the appliance.

- **data server** identifies the database server on a Teradata appliance as defined in the hosts file on the client. This data server is the same entry that you usually specify in the SERVER= entry of a LIBNAME statement for Teradata. For more information about specifying the LIBNAME statements for Teradata and other relational databases, see the SAS/ACCESS Interface documentation for the specific database.

The key variable that determines whether the OPTGRAPH procedure executes in distributed mode is the grid host. If no grid host is specified, PROC OPTGRAPH runs in single-machine mode on the client. The
installation location and data server are needed to ensure that a connection to the grid host can be made.
Specifying a data server is necessary only on a Teradata appliance and depends on the entries in the client
hosts file. The hosts file specifies the server (with a suffix that consists of “cop” and a number) and an IP
address. For example:

```
myservercop1 33.44.55.66
```

You can set an environment variable directly from the SAS program by using the OPTION SET= command.
The following example shows the grid host and installation location options as they might be specified for a
Greenplum appliance (no data server option is needed):

```
option set=GRIDHOST  ="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
```

Similarly, the following example shows the grid host and installation location options as they might be
specified for a Teradata appliance:

```
option set=GRIDHOST  ="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
```

Alternatively, you can set the parameters in the PERFORMANCE statement in the high-performance
procedure. For example:

```
performance host      ="grid001.example.com"
          install   ="/opt/TKGrid";
```

A specification in the PERFORMANCE statement overrides a specification of an environment variable
without resetting its value. An environment variable that you set in the SAS session with an OPTION SET=
command remains in effect until it is modified or until the SAS session terminates.

**PERFORMANCE Statement**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed comput-
ing, passes variables that describe the distributed computing environment, and requests detailed results about
the performance characteristics of a high-performance procedure. You can also use the PERFORMANCE
statement to control whether the procedure executes in single-machine mode or distributed mode. This
chapter uses the PERFORMANCE statement in both the OPTGRAPH procedure and the HPDS2 procedure.

You can specify the following performance-options in the PERFORMANCE statement.

**COMMIT=number**

requests that the procedure write periodic messages to the SAS log when the number of observations
that are sent from the client to the appliance for distributed processing exceeds an integer multiple of
number.

PROC OPTGRAPH ignores this option when it runs alongside the database, because no observations
are sent from the client to the appliance.
However, this option is useful when you use PROC HPDS2 to distribute data to the appliance. PROC HPDS2 sends the data in blocks to the appliance. Whenever the number of observations sent exceeds an integer multiple of `number`, a SAS log message is generated. The message indicates the actual number of observations distributed, not an integer multiple of `number`.

**DETAILS**
requests a table that shows a timing breakdown of the procedure steps.

**HOST=** "name"

**GRIDHOST=** "name"
specifies the name of the appliance host in single or double quotation marks. This option overrides the value of the GRIDHOST environment variable.

**INSTALL=** "name"

**INSTALLLOC=** "name"
specifies the name (in single or double quotation marks) of the directory in which libraries that are shared by SAS high-performance software are installed on the appliance. This option overrides the value of the GRIDINSTALLLOC environment variable.

**NTHREADS=** `number`

specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, the number of threads is determined by the number of CPUs on the host on which the analytic computations execute.

By default, the OPTGRAPH procedure executes in multiple concurrent threads unless the NOTHREADS system option is specified or unless you force single-threaded execution by specifying NTHREADS=1. The largest value that you can specify for `number` is 256.

**NOTE:** The SAS system option THREADS | NOTHREADS applies to the client machine on which the OPTGRAPH procedure executes. It does not apply to the compute nodes in a distributed environment.

**TIMEOUT=** `s`
specifies the time-out in seconds for the procedure to wait for a connection to the appliance and establish a connection back to the client. The default is 120 seconds. If jobs are submitted to the appliance through workload management tools that might suspend access to the appliance for a longer period, you might want to increase the time-out value.

---

**Distributing Input Data to the Appliance**

As described in the section “Alongside-the-Database Distributed Mode” on page 4, the OPTGRAPH procedure in distributed mode supports the alongside-the-database model of execution.

To run community detection in distributed mode, you should distribute the links data set by the column that represents the `from` node before you invoke PROC OPTGRAPH. If the links data are not distributed in this way, PROC OPTGRAPH automatically redistributes the links data internally before community detection. This could be time-consuming if the graph contains a large number of links.

Community detection in distributed mode requires a directed graph. If your graph is undirected, you must first convert it to a directed graph by repeating each link in the opposite direction.
To run centrality computations in distributed mode, the links data set must contain an additional column, `cluster`, which identifies the community that corresponds to the `from` node and `to` node for each link. PROC OPTGRAPH does not require the links data set to be distributed by the `cluster` column before running centrality computations. This is because PROC OPTGRAPH redistributes the links data internally before centrality computation if it detects that the data set is not distributed by the `cluster` column.

For more information about how to distribute a data set to each database, see *SAS/ACCESS for Relational Databases: Reference*.

---

### Community Detection

Community detection partitions a graph into communities such that the links within the community subgraphs are more densely connected than the links between communities.

You use the `COMMUNITY` statement to detect communities. The following section describes the options for this statement. For more information about community detection, see the section “Community Detection Details” on page 10.

---

#### COMMUNITY Statement

```
COMMUNITY < options > ;
```

You can specify the following `options` in the `COMMUNITY` statement when running PROC OPTGRAPH in distributed mode.

- **ALGORITHM=PARALLEL_LABEL_PROP**
  specifies which algorithm to use. Currently, only the parallel label propagation algorithm (PARALLEL_LABEL_PROP) is supported in distributed mode.

- **LOGLEVEL=**
  controls the amount of information that is displayed in the SAS log. Table 1.1 describes the valid values for this option.

<table>
<thead>
<tr>
<th>number</th>
<th>string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NONE</td>
<td>Turns off all algorithm-related messages in the SAS log</td>
</tr>
<tr>
<td>1</td>
<td>BASIC</td>
<td>Displays a basic summary of the algorithmic processing</td>
</tr>
<tr>
<td>2</td>
<td>MODERATE</td>
<td>Displays a summary of the algorithmic processing</td>
</tr>
<tr>
<td>3</td>
<td>AGGRESSIVE</td>
<td>Displays a detailed summary of the algorithmic processing</td>
</tr>
</tbody>
</table>

  The default is the value that you specify in the `LOGLEVEL=` option in the PROC OPTGRAPH statement (or BASIC if that option is not specified).

- **MAXITER=**
  specifies the maximum number of iterations that the algorithm allows. The default is 100.
OUT_COMM_LINKS=SAS-data-set
specifies the output data set that describes the links between communities.

OUT_COMMUNITY=SAS-data-set
specifies the output data set that contains the number of nodes in each community.

OUT_INTRA_COMM_LINKS=SAS-data-set
specifies the output data set that describes the links within each community.

OUT_LEVEL=SAS-data-set
specifies the output data set that contains community information at different resolution levels.

OUT_OVERLAP=SAS-data-set
specifies the output data set that describes the intensity of each node.

RANDOM_FACTOR=number
specifies the random factor for the parallel label propagation algorithm. Specify a number between 0 and 1. At each iteration, number \times 100\% of the nodes are randomly selected to skip the label propagation step. The default is 0.15, which means that 15\% of the nodes skip the label propagation step at each iteration.

RANDOM_SEED=number
specifies the initial seed for random number generation used in the parallel label propagation algorithm. At each iteration, some nodes are randomly selected to skip the label propagation step, based on the value that you specify in the RANDOM_FACTOR= option. To change the sequence of random numbers generated by changing the initial seed, specify a number in the RANDOM_SEED= option. The default is 1234.

RECURSIVE(MAX_COMM_SIZE=number)
requests that the algorithm recursively break down large communities into smaller ones until all communities have a size that is less than or equal to number. This option starts with the keyword RECURSIVE, followed by the MAX_COMM_SIZE= suboption enclosed in parentheses—for example, RECURSIVE (MAX_COMM_SIZE=200). MAX_COMM_SIZE= specifies the maximum number of nodes to be contained in any community.

For information about using the RECURSIVE (MAX_COMM_SIZE=) option, see the section “Large Community” on page 10.

RESOLUTION_LIST=number_list
specifies a list of resolution values that are separated by spaces (for example, 1.0 0.6 0.2). Multiple resolution values enable you to run community detection multiple times, each time with a different resolution value. Valid values are any nonnegative numbers; the default is 0.001.

For more information about using the RESOLUTION_LIST= option, see the section “Large Community” on page 10.

TOLERANCE=number
stops iterations when the percentage of label changes for all nodes in the graph falls within the tolerance specified by number. The valid range is between 0 and 1. The default is 0.01.
Community Detection Details

Community detection partitions a graph into communities such that the links within the community subgraphs are more densely connected than the links between communities. You use the COMMUNITY statement to detect communities. The options for this statement are described in the section “COMMUNITY Statement” on page 8.

In distributed mode, PROC OPTGRAPH implements the parallel label propagation algorithm, which was developed at SAS. The goal is to move a node to a community to which most of its neighbors belong. Briefly, the parallel label propagation algorithm does the following:

1. It initializes each node as its own community.

2. At each iteration, it randomly chooses some nodes as candidates and moves each candidate from its current community to the neighboring community that has the most nodes. It repeats this step until the number of movements is smaller than the specified tolerance value or the maximum number of iterations has been reached.

The parallel label propagation algorithm is an extension of the synchronous label propagation algorithm proposed in Raghavan, Albert, and Kumara (2007). During each iteration, nodes update their labels simultaneously by using the node label information from the previous iteration. In this approach, node labels can be updated in parallel. However, simultaneous updating of this nature often leads to oscillating labels because of the bipartite subgraph structure often present in large graphs. To address this issue, at each iteration the parallel algorithm skips the labeling step at some randomly chosen nodes in order to break the bipartite structure. You can control the random samples that the algorithm takes by specifying the RANDOM_FACTOR= and RANDOM_SEED= options in the COMMUNITY statement.

As you can see from the description, the algorithm adopts a heuristic local optimization approach. The final result often depends on the sequence of nodes that are presented in the links input data set. Therefore, if the sequence of nodes in the links data set changes, the result is likely to be different.

In distributed mode, the link data must be a directed graph, and you must predistribute the graph to the appliance according to the from column values before you call PROC OPTGRAPH, as described in the section “Distributing Input Data to the Appliance” on page 7. If the original data are an undirected graph, you must convert the undirected graph to a directed graph by replacing each undirected link $A \leftrightarrow B$ with two directed links: $A \rightarrow B$ and $B \rightarrow A$. This ensures that each node can access its immediate neighbors on all computing nodes, thus minimizing data movement among computing nodes.

Large Community

It has often been observed in practice that the number of nodes contained in communities (produced by community detection algorithms) usually follows a power law distribution. That is, a few communities contain a very large number of nodes, whereas most communities contain a small number of nodes. This is especially true for large graphs. PROC OPTGRAPH provides two approaches for you to alleviate this problem:

- You can use the RECURSIVE option to recursively break large communities into smaller ones. At the first step, PROC OPTGRAPH processes data as if no RECURSIVE option were specified. At the end
of this step, it checks whether the community result has reached the maximum community size. If the community result has reached the maximum community size, PROC OPTGRAPH stops iterations and outputs results. Otherwise, it treats each large community as an independent graph and recursively applies community detection on top of it.

In certain cases, a community is not further split even if it does not meet the maximum community size that you specify. One example is a star-shaped community that contains 200 nodes when MAX_COMM_SIZE is specified as 100.

- You can use the RESOLUTION_LIST= option to assign a different value from the default value of 0.001. The resolution value that is specified in this option can be interpreted as the minimal density of communities for an undirected and unweighted graph. The density of a community is defined as the number of links inside the community divided by the total number of possible links. A larger resolution value is likely to result in communities that contain fewer nodes. For more information about resolution values for label propagation, see Traag, Van Dooren, and Nesterov (2011).

If you supply multiple resolution values at one time, the OPTGRAPH procedure performs community detection multiple times, each time with a different resolution value. This is equivalent to calling the OPTGRAPH procedure several times, each time with a different (single) resolution value specified in the RESOLUTION_LIST= option.

The value that you specify in the RESOLUTION_LIST= option has a major impact on the running time of the algorithm. When you specify a large resolution value, the algorithm is likely to create many tiny communities, and nodes are likely to change communities between iterations. Therefore the algorithm might not converge properly. On the other hand, when you specify a small resolution value, the algorithm might find some very large communities, such as a community that contains more than one million nodes. In this case, if you specify the RECURSIVE option, the algorithm spends a long time in the recursive step in order to break large communities into smaller ones.

The recommended approach is to first experiment with a set of resolution values without using the RECURSIVE option. At the end of the run, examine the resulting modularity values and the community size distributions. Remove the resolution values that lead to small modularity values or huge communities. Then add the RECURSIVE option to the COMMUNITY statement, if desired, and run PROC OPTGRAPH again.

**Output Data Sets**

Community detection produces up to six output data sets. In these data sets, resolution level numbers are in the same order as the values that are specified in the RESOLUTION_LIST= option. For example, if RESOLUTION_LIST=0.001 0.005 0.01, then resolution level 1 is at value 0.001, resolution level 2 is at value 0.005, and resolution level 3 is at value 0.01.

**OUT_NODES= Data Set**

This data set describes the community identifier of each node. If multiple resolution values have been specified, the data set reports the community identifier of each node at each resolution level. The data set contains the following columns:

- **node**: node label

- **community_i**: community identifier at resolution level i, where i is the resolution level number as previously described. There are K such columns if K different values are specified in the RESOLUTION_LIST= option.
**OUT_LEVEL= Data Set**
This data set describes the number of communities and their corresponding modularity values at various resolution levels. It contains the following columns:

- level: resolution level number
- resolution: resolution value
- communities: number of communities at the current resolution level
- modularity: modularity value at the current resolution level

**OUT_COMMUNITY= Data Set**
This data set describes the number of nodes in each community. It contains the following columns:

- level: resolution level number
- resolution: resolution value
- community: community identifier
- nodes: number of nodes contained in the community

**OUT_OVERLAP= Data Set**
This data set describes the intensity of each node. At the end of community detection, a node could have links that connect to multiple communities. The intensity of a node is computed as the sum of the link weights that connect to nodes in the specified community divided by the total link weights of the node. This data set is computationally expensive to produce, and it requires a large amount of disk space. Therefore, this data set is not produced if you specify multiple resolution values in the RESOLUTION_LIST= option and ALGORITHM=PARALLEL_LABEL_PROP.

The data set contains the following columns:

- node: node label
- community: community identifier
- intensity: intensity of the node that belongs to the community

**OUT_COMM_LINKS= Data Set**
This data set describes how communities are connected. This data set is computationally expensive to produce, and it requires a large amount of disk space. Therefore, this data set is not produced if you specify multiple resolution values in the RESOLUTION_LIST= option.

The data set contains the following columns:

- from_community: community identifier of the from community
- to_community: community identifier of the to community
- link_weight: sum of link weights of all links between from_community and to_community

**OUT_INTRA_COMM_LINKS= Data Set**

This data set describes how the nodes are connected within each community. This data set is computationally expensive to produce, and it requires a large amount of disk space. Therefore, this data set is not produced if you specify multiple resolution values in the RESOLUTION_LIST= option.

The data set contains the following columns:

- cluster: the cluster ID (that is, the community ID)
- from: the node label of the from node
- to: the node label of the to node
- weight: the link weight between from and to
- weight2: the second link weight between from and to

The column weight2 is created if the input links data set has a weight2 column, even though community detection does not use this column during the computation. This is because the OUT_INTRA_COMM_LINKS= data set is used as input to the centrality computation step, in which PROC OPTGRAPH might need two link weight columns: one column for computing PageRank, eigenvector, and hub/authority centralities, and the other column for computing betweenness and closeness centralities. Therefore, community detection carries the second link weight column to the output.

---

**Example: Community Detection on a Simple Undirected Graph**

This section illustrates the use of the community detection algorithm in distributed mode on the simple undirected graph depicted in Figure 1.1.

**Figure 1.1** A Simple Undirected Graph
The following statements create the data set LinkSetIn:

```sas
data LinkSetIn;
  input from $ to $ @@;
datalines;
A B A C A D B C C D
C E D F F G F H F I
G H G I I J J K J L
K L
;
```

Before you run community detection in distributed mode, you must convert the undirected graph to a directed graph by replicating links in both directions, as shown in the following statements:

```sas
data LinkSetIn;
  set LinkSetIn;
  output;
  tmp = from;
  from = to;
  to = tmp;
  output;
  drop tmp;
run;
```

Then you can use PROC HPDS2 to distribute the links data to the appliance by from as follows:

```sas
libname gplib greenplm
  server = "grid001.example.com"
  schema = public
  user = dbuser
  password = dbpass
  database = hps
  preserve_col_names=yes;

proc datasets nolist lib=gplib;
  delete LinkSetIn;
quit;

proc hpds2
  data = LinkSetIn
  out = gplib.LinkSetIn (distributed_by='distributed by (from)');
  performance
    host = "grid001.example.com"
    install = "/opt/TKGrid"
  data DS2GTF.out;
    method run();
      set DS2GTF.in;
    end;
    enddata;
  run;
```

The LIBNAME statement option PRESERVE_COL_NAMES=YES is used because the links data set contains the variable from, which is a reserved keyword for DBMS tables that use SAS/ACCESS. For more information, see SAS/ACCESS for Relational Databases: Reference.
After the data have been distributed to the appliance, you can invoke PROC OPTGRAPH. In this example, some of the output data sets are written to the appliance and other output data sets are written to the client machine. It is recommended that the large output data sets be stored in the DBMS, because transferring data from the appliance to the client machine can take a very long time. In addition, further analysis (such as centrality computation) can benefit from the data’s already being distributed on the appliance. In the following statements, CommLevelOut and CommOut are stored on the client machine in the WORK library, and the remaining output data sets are stored on the appliance, although the output data sets for this simple example are very small:

```plaintext
proc datasets nolist lib=gplib;
   delete NodeSetOut CommOverlapOut CommLinksOut CommIntraLinksOut;
quit;

proc optgraph
   graph_direction = directed
   data_links = gplib.LinkSetIn
   out_nodes = gplib.NodeSetOut;
   performance
      host = "grid001.example.com"
      install = "/opt/TKGrid";
   community
      resolution_list = 0.001
      algorithm = parallel_label_prop
      out_level = CommLevelOut
      out_community = CommOut
      out_overlap = gplib.CommOverlapOut
      out_comm_links = gplib.CommLinksOut
      out_intra_comm_links = gplib.CommIntraLinksOut;
run;
```

The data set NodeSetOut contains the community identifier of each node. It is shown in Figure 1.2.

**Figure 1.2** Community Nodes Output

<table>
<thead>
<tr>
<th>node</th>
<th>community</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The data set CommLevelOut contains the number of communities and the corresponding modularity values that are found at each resolution level. It is shown in Figure 1.3.
The data set CommOut contains the number of nodes that are contained in each community. It is shown in Figure 1.4.

The data set CommOverlapOut contains the intensity of each node that belongs to multiple communities. It is shown in Figure 1.5. In this example, node F is connected to two communities, with 75% of its links connecting to community 1 and 25% of its links connecting to community 2.

The data set CommLinksOut shows how the communities are interconnected. It is shown in Figure 1.6.
The data set CommIntraLinksOut shows how the nodes are connected within each community. It is shown in Figure 1.7.

**Figure 1.7** Intracommunity Links Output

<table>
<thead>
<tr>
<th>cluster</th>
<th>from</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>I</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>I</td>
</tr>
<tr>
<td>1</td>
<td>J</td>
<td>K</td>
</tr>
<tr>
<td>1</td>
<td>J</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>K</td>
<td>J</td>
</tr>
<tr>
<td>1</td>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>J</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>K</td>
</tr>
<tr>
<td>0</td>
<td>E</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

---

**Centrality Computation by Cluster**

The *centrality* of a node in a graph indicates its relative importance within a graph. In the field of network analysis, many different types of centrality metrics are used to better understand levels of prominence. For more information, see the section “Centrality” in *SAS OPTGRAPH Procedure: Graph Algorithms and Network Analysis*.

When running in distributed mode, you can use the CENTRALITY statement in PROC OPTGRAPH along with the BY_CLUSTER option to process the induced subgraphs that are defined by the output of the community detection algorithm or to process the induced subgraphs that are defined by any general partition of the links in the graph. The typical use case of the BY_CLUSTER option is described in the section “Processing by Cluster” in *SAS OPTGRAPH Procedure: Graph Algorithms and Network Analysis*. The main
difference when you run in distributed mode is that the cluster variable is defined in the links input data set (which corresponds to the DATA_LINKS= option), not in the nodes input data set (which corresponds to the DATA_NODES= option). In distributed mode, there is no need for the DATA_NODES= option.

As in the process described previously for community detection, you can predistribute the links data set to the grid by cluster by using one of the methods described in the section “Distributing Input Data to the Appliance” on page 7. However, as mentioned in the section “Recommended Workflow” on page 4, the recommended workflow is to use the OUT_INTRA_COMM_LINKS= output data set that results from running community detection as input to the centrality algorithm. This data set already contains the cluster variable, which identifies the assigned community for each link, and the data set is stored on the appliance.

The following sections provide two examples of running centrality in distributed mode. The first example shows how to use the output from running community detection as input to the centrality algorithm. It then shows an alternative manual process of predistributing the data by cluster, to be used as input to the centrality algorithm. In the second example, the cluster variable does not need to come from the community detection algorithm, but it can represent any partition of the graph.

### Example: Centrality by Community for a Simple Undirected Graph

In “Example: Community Detection on a Simple Undirected Graph” on page 13, the COMMUNITY statement in PROC OPTGRAPH is used to detect communities in a simple undirected graph. The OUT_INTRA_COMM_LINKS= option in that example stores the resulting community partition in a table called gplib.CommIntraLinksOut. This table is also shown in Figure 1.7 on page 17.

Because this data set is stored on the appliance, it is already in the appropriate form to be used as input for running centrality by cluster. You can calculate the centrality metrics on the appliance, in parallel, by using this data set as input, as in the following call to PROC OPTGRAPH:

```plaintext
proc datasets nolist lib=gplib;
    delete NodeSetCentrality;
run;

proc optgraph
    data_links = gplib.CommIntraLinksOut
    out_nodes = gplib.NodeSetCentrality;
performance
    host = "grid001.example.com"
    install = "/opt/TKGrid";
centrality
    by_cluster
    degree = out
    influence = unweight
    close = unweight
    between = unweight
    eigen = unweight;
run;
```

If you use the library gplib along with the OUT_NODES= option, the results of centrality computations are also stored in distributed form on the appliance in gplib.NodeSetCentrality. For the sake of display, a local version of the data is created and sorted as follows:
Example: Centrality by Cluster for a Simple Undirected Graph

```sas
data NodeSetCentrality;
  set gplib.NodeSetCentrality;
run;
proc sort data=NodeSetCentrality;
  by cluster descending centr_eigen_unwt ;
run;
```

The results are shown in Figure 1.8.

**Figure 1.8** Centrality for All Induced Communities

<table>
<thead>
<tr>
<th>node</th>
<th>cluster</th>
<th>centr_degree_out</th>
<th>centr_eigen_unwt</th>
<th>centr_close_unwt</th>
<th>centr_between_unwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>4</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.58333</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>0.89897</td>
<td>0.80000</td>
<td>0.08333</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0.70711</td>
<td>0.66667</td>
<td>0.00000</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>2</td>
<td>0.70711</td>
<td>0.66667</td>
<td>0.00000</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0.37236</td>
<td>0.57143</td>
<td>0.00000</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>2</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>2</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>2</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>3</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.16667</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>3</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.16667</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>2</td>
<td>0.78078</td>
<td>0.75000</td>
<td>0.00000</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>2</td>
<td>0.78078</td>
<td>0.75000</td>
<td>0.00000</td>
</tr>
<tr>
<td>centr_influence1_unwt</td>
<td>centr_influence2_unwt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.80000</td>
<td>1.60000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.60000</td>
<td>1.60000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.40000</td>
<td>1.40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.40000</td>
<td>1.40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.20000</td>
<td>0.80000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.66667</td>
<td>1.33333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.66667</td>
<td>1.33333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.66667</td>
<td>1.33333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.75000</td>
<td>1.75000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.75000</td>
<td>1.75000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.50000</td>
<td>1.50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.50000</td>
<td>1.50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For information about other options in the CENTRALITY statement, see the section “Centrality” in SAS OPTGRAPH Procedure: Graph Algorithms and Network Analysis.

**Example: Centrality by Cluster for a Simple Undirected Graph**

This example uses the same simple undirected graph as in the previous example; it is shown again in Figure 1.9. However, this example does not use community detection. Instead, the data set is manually predistributed by the cluster variable, where the cluster variable can define any partition of the nodes.
The following statements create the data set LinkSetIn:

```plaintext
data LinkSetIn;
   input from $ to $ @@;
   datalines;
   A B A C A D B C C D
   C E D F F G F H F I
   G H G I I J J K J L
   K L
;
```

The graph seems to have three distinct parts, which are connected by just a few links. Assume that you have already partitioned the data set into three sets of nodes: 

- \( N^0 = \{A, B, C, D, E\} \)
- \( N^1 = \{F, G, H, I\} \)
- \( N^2 = \{J, K, L\} \)

The induced subgraphs on these three sets of nodes are shown in blue in Figure 1.10 through Figure 1.12.
The following data sets define the three induced subgraphs:

```plaintext
data LinkSetIn0;
  input from $ to $ @@;
  datalines;
A B A C A D B C C D C E
;  
data LinkSetIn1;
  input from $ to $ @@;
  datalines;
F G F H F I G H G I
;  
data LinkSetIn2;
  input from $ to $ @@;
  datalines;
J K J L K L
;  
```
To calculate centrality metrics on the three subgraphs, you could run PROC OPTGRAPH three times, as follows:

```sas
proc optgraph
data_links = LinkSetIn0
out_nodes = NodeSetOut0;
centrality
degree = out
influence = unweight
close = unweight
between = unweight
eigen = unweight;
run;
```

```sas
proc optgraph
data_links = LinkSetIn1
out_nodes = NodeSetOut1;
centrality
degree = out
influence = unweight
close = unweight
between = unweight
eigen = unweight;
run;
```

```sas
proc optgraph
data_links = LinkSetIn2
out_nodes = NodeSetOut2;
centrality
degree = out
influence = unweight
close = unweight
between = unweight
eigen = unweight;
run;
```

This produces the results shown in Figure 1.13 through Figure 1.15.

**Figure 1.13  Centrality for Induced Subgraph 0**

<table>
<thead>
<tr>
<th>node</th>
<th>centr_degree_out</th>
<th>centr_eigen_unwt</th>
<th>centr_close_unwt</th>
<th>centr_between_unwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>0.89897</td>
<td>0.80000</td>
<td>0.08333</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0.70711</td>
<td>0.66667</td>
<td>0.00000</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.58333</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>0.70711</td>
<td>0.66667</td>
<td>0.00000</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>0.37236</td>
<td>0.57143</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>centr_influence1_unwt</th>
<th>centr_influence2_unwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>1.6</td>
</tr>
<tr>
<td>0.4</td>
<td>1.4</td>
</tr>
<tr>
<td>0.8</td>
<td>1.6</td>
</tr>
<tr>
<td>0.4</td>
<td>1.4</td>
</tr>
<tr>
<td>0.2</td>
<td>0.8</td>
</tr>
</tbody>
</table>
A much more efficient way to process these graphs is to use the BY_CLUSTER option. The section “Processing by Cluster” in SAS OPTGRAPH Procedure: Graph Algorithms and Network Analysis shows how to use the BY_CLUSTER option for running in single-machine mode. This example shows the same process for running in distributed mode.

Define the partitions of the original graph by adding a cluster variable to the links data set. This variable denotes the partition to which each link belongs. If the partition is defined over nodes, then any links that span from one partition to another are removed from the input data set.

```sas
data LinkSetCluster;
  input from $ to $ cluster @@;
  datalines;
  A B 0 A C 0 A D 0 B C 0 C D 0 C E 0
  F G 1 F H 1 F I 1 G H 1 G I 1
  J K 2 J L 2 K L 2
;```

Next, use PROC HPDS2 to distribute the links data set to the appliance by cluster, as follows:

```
libname gplib greenplm
    server = "grid001.example.com"
    schema = public
    user = dbuser
    password = dbpass
    database = hps
    preserve_col_names = yes;

proc datasets nolist lib=gplib;
    delete LinkSetIn;
run;

proc hpds2
    data = LinkSetCluster
    out = gplib.LinkSetIn (distributed_by='distributed by (cluster)');
    performance
        host = "grid001.example.com"
        install = "/opt/TKGrid"
    data DS2GTF.out;
    method run();
        set DS2GTF.in;
    end;
enddata;
run;
```

You use the LIBNAME option PRESERVE_COL_NAMES=YES because the links data set contains the variable **from**, which is a keyword reserved for DBMS tables that use SAS/ACCESS. (See SAS/ACCESS for Relational Databases: Reference.)

Now, by using one call to PROC OPTGRAPH, you can process all three induced subgraphs on the appliance in parallel, as follows:

```
proc datasets nolist lib=gplib;
    delete NodeSetCentrality;
run;

proc optgraph
    data_links = gplib.LinkSetCluster
    out_nodes = gplib.NodeSetCentrality;
    performance
        host = "grid001.example.com"
        install = "/opt/TKGrid"
    centrality
        by_cluster
            degree = out
            influence = unweight
            close = unweight
            between = unweight
            eigen = unweight;
run;
```

In this example, the results in the data set that is specified by the OUT_NODES= option are stored in distributed form on the appliance in **gplib.NodeSetCentrality**. For the sake of display, a local version of the data is created and sorted as follows:
data NodeSetCentrality;
  set gplib.NodeSetCentrality;
run;
proc sort data=NodeSetCentrality;
  by cluster descending centr_eigen_unwt;
run;

The results are shown in Figure 1.16.

**Figure 1.16** Centrality for All Induced Subgraphs

<table>
<thead>
<tr>
<th>node</th>
<th>cluster</th>
<th>centr_degree_out</th>
<th>centr_eigen_unwt</th>
<th>centr_close_unwt</th>
<th>centr_between_unwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>4</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.58333</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>0.89897</td>
<td>0.80000</td>
<td>0.08333</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>0.70711</td>
<td>0.66667</td>
<td>0.00000</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>2</td>
<td>0.70711</td>
<td>0.66667</td>
<td>0.00000</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0.37236</td>
<td>0.57143</td>
<td>0.00000</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>2</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>2</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>2</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>3</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.16667</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>3</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.16667</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>2</td>
<td>0.78078</td>
<td>0.75000</td>
<td>0.00000</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>2</td>
<td>0.78078</td>
<td>0.75000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>centr_influence1_unwt</th>
<th>centr_influence2_unwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.80000</td>
<td>1.60000</td>
</tr>
<tr>
<td>0.60000</td>
<td>1.60000</td>
</tr>
<tr>
<td>0.40000</td>
<td>1.40000</td>
</tr>
<tr>
<td>0.40000</td>
<td>1.40000</td>
</tr>
<tr>
<td>0.20000</td>
<td>0.80000</td>
</tr>
<tr>
<td>0.66667</td>
<td>1.33333</td>
</tr>
<tr>
<td>0.66667</td>
<td>1.33333</td>
</tr>
<tr>
<td>0.66667</td>
<td>1.33333</td>
</tr>
<tr>
<td>0.75000</td>
<td>1.75000</td>
</tr>
<tr>
<td>0.75000</td>
<td>1.75000</td>
</tr>
<tr>
<td>0.50000</td>
<td>1.50000</td>
</tr>
<tr>
<td>0.50000</td>
<td>1.50000</td>
</tr>
</tbody>
</table>

**References**


Index

ALGORITHM= option
   COMMUNITY statement, 8

COMMUNITY statement
   statement options, 8

DETAILS option
   PERFORMANCE statement, 7

HOST= option
   PERFORMANCE statement, 7

INSTALL= option
   PERFORMANCE statement, 7

LOGLEVEL= option
   COMMUNITY statement, 8

MAXITER= option
   COMMUNITY statement, 8

NTHREADS= option
   PERFORMANCE statement, 7

OUT_COMM_LINKS= option
   COMMUNITY statement, 9

OUT_COMMUNITY= option
   COMMUNITY statement, 9

OUT_INTRA_COMM_LINKS= option
   COMMUNITY statement, 9

OUT_LEVEL= option
   COMMUNITY statement, 9

OUT_OVERLAP= option
   COMMUNITY statement, 9

PERFORMANCE statement
   statement options, 6

RANDOM_FACTOR= option
   COMMUNITY statement, 9

RANDOM_SEED= option
   COMMUNITY statement, 9

RECURSIVE(MAX_COMM_SIZE=) option
   COMMUNITY statement, 9

RESOLUTION_LIST= option
   COMMUNITY statement, 9

TIMEOUT= option
   PERFORMANCE statement, 7

TOLERANCE= option
   COMMUNITY statement, 9
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.