SAS® OPTGRAPH Procedure 13.1

High-Performance Features

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® OPTGRAPH Procedure 13.1: High-Performance Features*. Cary, NC: SAS Institute Inc.

SAS® OPTGRAPH Procedure 13.1: High-Performance Features

Copyright © 2013, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

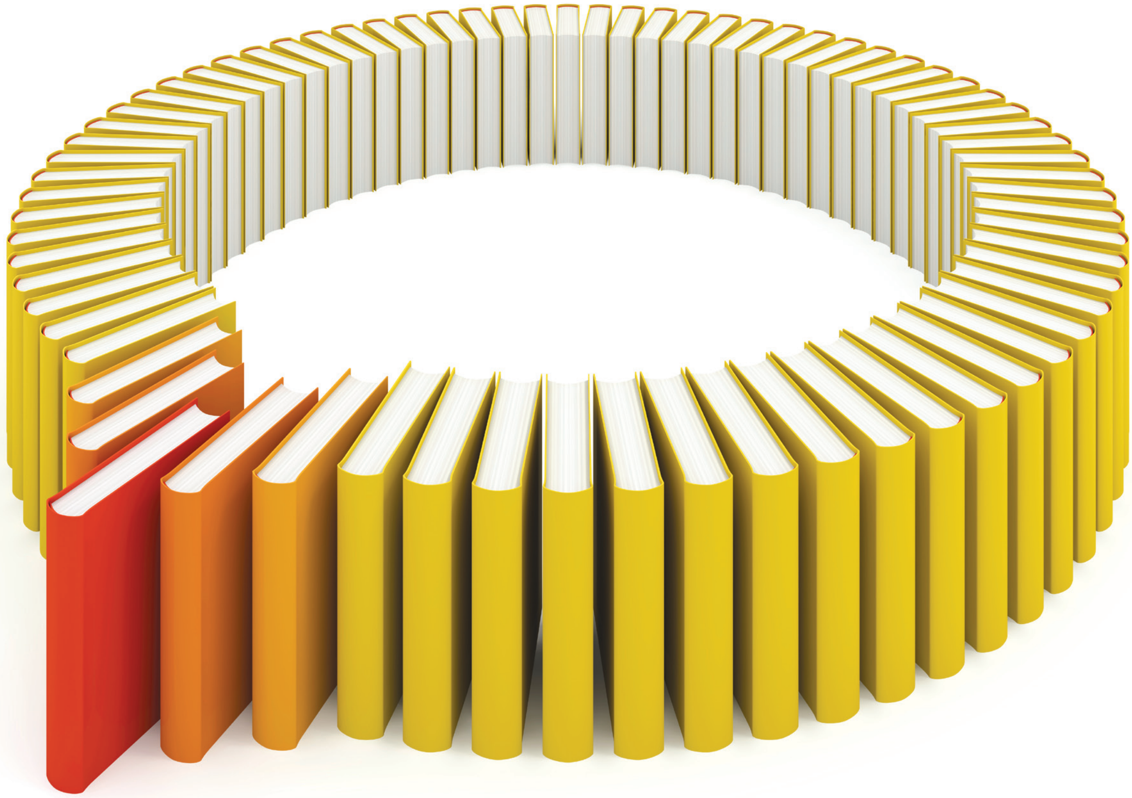
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

December 2013

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our offerings, visit **support.sas.com/bookstore** or call 1-800-727-3228.

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.



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.

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. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613

Contents

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

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 **8**
  - Distribute Data to Greenplum 9
  - Distribute Data to Teradata 10
- Community Detection **10**
  - COMMUNITY Statement 11
  - Community Detection Details 12
  - Example: Community Detection on a Simple Undirected Graph 16
- Centrality Computation by Cluster **20**
  - Example: Centrality by Community for a Simple Undirected Graph 21
  - Example: Centrality by Cluster for a Simple Undirected Graph 23
- References **28**

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.

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 8. The links data set must be distributed by the variable that represents the *from* node of each link.

2. Run PROC OPTGRAPH with the COMMUNITY statement to perform community detection as described in the section "Community Detection" on page 10. Write the OUT_INTRA_COMM_LINKS= output data set from community detection to the appliance, where it is distributed by cluster, the community identifier that corresponds to the assigned community for the *from* and *to* node for each link.

   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 10.

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 20. As the links input data set, use the OUT_INTRA_COMM_LINKS= output data set that was created in step 2; it is already distributed by cluster.

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 can run PROC OPTGRAPH alongside the database, you must distribute the data to the appliance. This step is described in the section "Distributing Input Data to the Appliance" on page 8. 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, installation location, and data server options as they might be specified for a Teradata appliance:

```
option set=GRIDHOST      ="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
option set=GRIDDATASERVER="td0001";
```

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"
            dataserver="td0001";
```

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 computing, 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, as described in the section "Distribute Data to Greenplum" on page 9. 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*.

DATASERVER=*"name"*

specifies the name (in single or double quotation marks) of the server in a Teradata appliance as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, assume that the *hosts* file defines the following as the server for Teradata:

```
myservercop1  33.44.55.66
```

Then a LIBNAME specification would be as follows:

```
libname tdlib teradata
   server     = "grid001.example.com"
   user       = dbuser
   password   = dbpass
   database   = hps;
```

The following PERFORMANCE statement causes the procedure to run alongside the Teradata server that is named grid001.example.com:

```
performance dataserver="grid001.example.com";
```

This option overrides the GRIDDATASERVER environment variable.

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. Before you can run PROC OPTGRAPH alongside the database, you must distribute the data to the appliance.

To run community detection in distributed mode, you must distribute the links data set. Community detection requires all links that have the same *from* node to be colocated on the same node of the appliance. Therefore, before running community detection, you must distribute the links data set by the column that represents the *from* node. 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, that identifies the community that corresponds to the *from* node and *to* node for each link, and the data set must be distributed by the cluster column. That is, all links within the same community must be colocated on the same node of the appliance. However, the cluster variable is determined as a result of running community detection in a previous call to PROC OPTGRAPH. That is, the output data set from community detection becomes the input data set for centrality computations. Therefore, if you specify that the output links data set from community detection be written directly to the appliance, you do not need to distribute this data set to run centrality computations. It is already distributed by the cluster variable.

The following sections describe how to distribute a data set to a Greenplum appliance and to a Teradata appliance. For more information about each data access engine, see *SAS/ACCESS for Relational Databases: Reference*.

Distribute Data to Greenplum

The following example shows how to use the HPDS2 procedure to copy a data set to a Greenplum database; the table is distributed by a column called from_node:

```
libname linkdata 'C:\mydata';

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

proc datasets nolist lib=gplib;
   delete links_data_123;
quit;

proc hpds2
   data = linkdata.links_data_123
   out  = gplib.links_data_123 (distributed_by='distributed by (from_node)');
   performance
      host    = "grid001.example.com"
      install = "/opt/TKGrid"
      commit  = 10000000;
   data DS2GTF.out;
      method run();
         set DS2GTF.in;
      end;
   enddata;
run;
```

If the output table links_data_123 already exists in the Greenplum database, the call to the DATASETS procedure removes the existing table from the database, because a DBMS usually does not support replacement operations on tables.

The OUT= option in the PROC HPDS2 statement specifies a table that uses the library gplib, which is the Greenplum library that is assigned in the second LIBNAME statement. This option also requests that PROC HPDS2 distribute the records by from_node among the data segments of the computing appliance. The statements that follow the PERFORMANCE statement are the DS2 program that copies the input data to the output data without further transformations.

For more information about the HPDS2 procedure, see *Base SAS Procedures Guide: High-Performance Procedures*.

Distribute Data to Teradata

The following example shows how to use a DATA step to copy a data set to a Teradata database; the table is distributed by a column called from_node:

```
libname linkdata 'C:\mydata';

libname tdlib teradata
   server     = "grid001.example.com"
   user       = dbuser
   password   = dbpass
   database   = hps;

proc datasets nolist lib=tdlib;
   delete links_data_123;
quit;

data tdlib.links_data_123 (bulkload=yes
                           dbcommit=10000000
                           DBCREATE_TABLE_OPTS="primary index(from_node)");
   set linkdata.links_data_123;
run;
```

If the output table links_data_123 already exists in the Teradata database, the call to PROC DATASETS removes the table from the database, because a DBMS usually does not support replacement operations on tables.

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 12.

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

Table 1.1 Values for LOGLEVEL= Option

number	*string*	**Description**
0	NONE	Turns off all algorithm-related messages in the SAS log
1	BASIC	Displays a basic summary of the algorithmic processing
2	MODERATE	Displays a summary of the algorithmic processing
3	AGGRESSIVE	Displays a detailed summary of the algorithmic processing

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=*number*
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 that belongs to multiple communities.

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 13.

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 13.

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 11.

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 8. If the original data are an undirected graph, you must convert the undirected graph to a directed graph by replacing each undirected link $A - B$ 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 a node that belongs to multiple communities. At the end of community detection, a node could have links that connect to multiple communities. The intensity of a node that belongs to community i is computed as the sum of the weights of links that connect community i 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.

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:

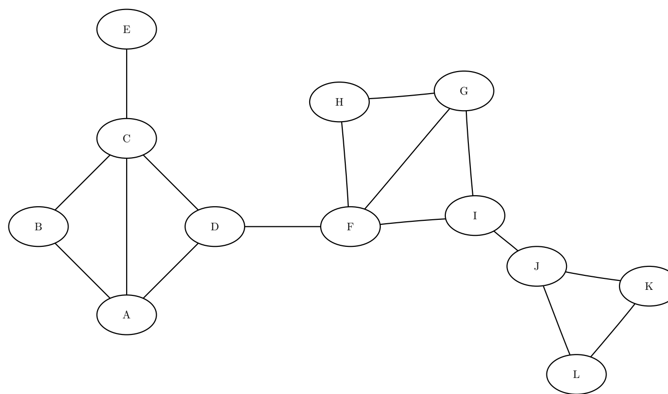
- cluster: the cluster ID (that is, the community ID)
- from: the node label of the *from* node
- to: the node label of the *to* node
- weight: the link weight between from and to
- weight2: the second link weight between from and to

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:

```
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:

```
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:

```
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:

```
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

node	community_1
E	2
B	2
C	2
D	2
A	2
J	0
K	0
L	0
H	1
I	1
F	1
G	1

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.

Figure 1.3 Community Level Summary Output

level	resolution	communities	modularity
0	.001	3	0.52148

The data set CommOut contains the number of nodes that are contained in each community. It is shown in Figure 1.4.

Figure 1.4 Community Number of Nodes Output

resolution	community	nodes
.001	0	3
.001	1	4
.001	2	5

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.

Figure 1.5 Community Overlap Output

node	community	intensity
E	2	1.00000
C	2	1.00000
B	2	1.00000
H	1	1.00000
G	1	1.00000
F	1	0.75000
F	2	0.25000
J	1	0.33333
J	0	0.66667
K	0	1.00000
D	2	0.66667
D	1	0.33333
A	2	1.00000
I	1	0.66667
I	0	0.33333
L	0	1.00000

The data set CommLinksOut shows how the communities are interconnected. It is shown in Figure 1.6.

Figure 1.6 Community Links Output

from_ community	to_community	link_ weight
0	1	1
2	1	1
1	0	1
1	2	1

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

cluster	from	to
1	H	F
1	H	G
1	I	F
1	I	G
1	F	H
1	F	I
1	F	G
1	G	H
1	G	I
1	G	F
2	E	C
2	B	A
2	B	C
2	C	A
2	C	B
2	C	D
2	C	E
2	D	A
2	D	C
2	A	B
2	A	C
2	A	D
0	J	K
0	J	L
0	K	J
0	K	L
0	L	J
0	L	K

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 8. 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 distributed by cluster.

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 16, 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, which is distributed by cluster on the appliance. This table is also shown in Figure 1.7 on page 20.

Because this data set is stored on the appliance and distributed by community (cluster), 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:

```
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:

```
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

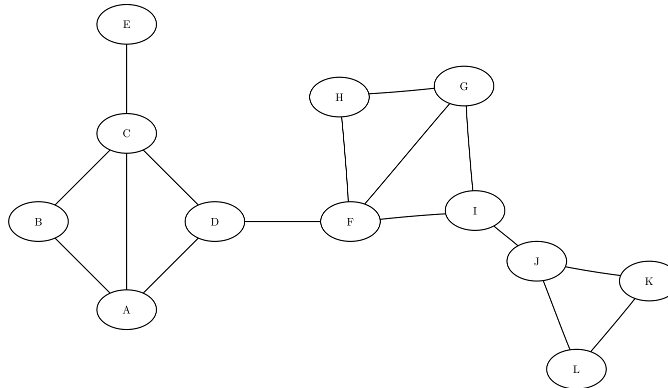
node	cluster	centr_degree_out	centr_eigen_unwt	centr_close_unwt	centr_between_unwt	centr_influence1_unwt	centr_influence2_unwt
K	0	2	1.00000	1.00000	0.00000	0.66667	1.33333
J	0	2	1.00000	1.00000	0.00000	0.66667	1.33333
L	0	2	1.00000	1.00000	0.00000	0.66667	1.33333
F	1	3	1.00000	1.00000	0.16667	0.75000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

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.

Figure 1.9 Undirected Graph



The following statements create the data set LinkSetIn:

```
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: $N^0 = \{A, B, C, D, E\}$, $N^1 = \{F, G, H, I\}$, and $N^2 = \{J, K, L\}$. The induced subgraphs on these three sets of nodes are shown in blue in Figure 1.10 through Figure 1.12.

Figure 1.10 Subgraph $N^0 = \{A, B, C, D, E\}$

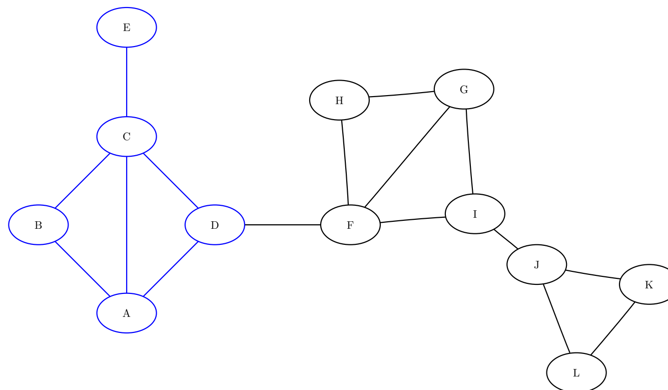


Figure 1.11 Subgraph $N^1 = \{F, G, H, I\}$

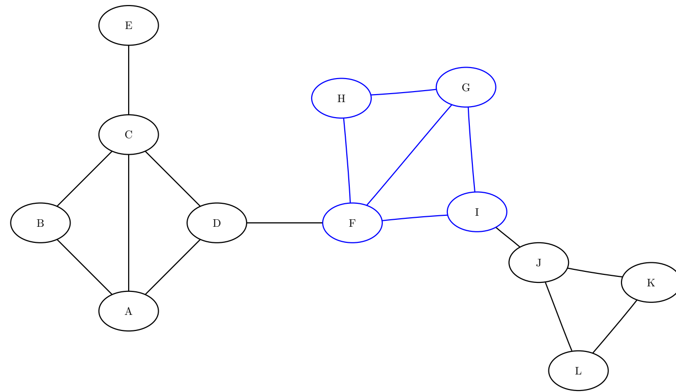
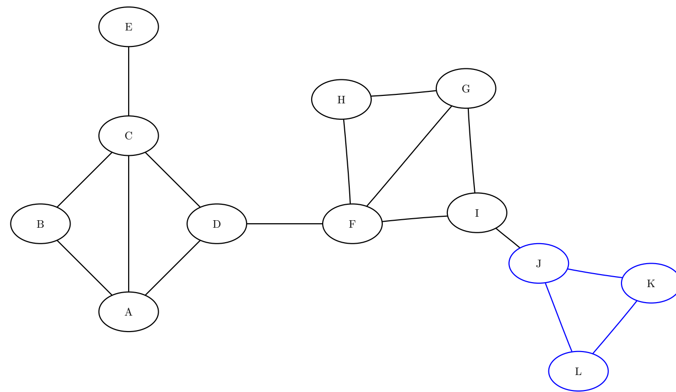


Figure 1.12 Subgraph $N^2 = \{J, K, L\}$



The following data sets define the three induced subgraphs:

```
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:

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

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

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

node	centr_ degree_ out	centr_ eigen_ unwt	centr_ close_ unwt	centr_ between_ unwt	centr_ influence1_ unwt	centr_ influence2_ unwt
A	3	0.89897	0.80000	0.08333	0.6	1.6
B	2	0.70711	0.66667	0.00000	0.4	1.4
C	4	1.00000	1.00000	0.58333	0.8	1.6
D	2	0.70711	0.66667	0.00000	0.4	1.4
E	1	0.37236	0.57143	0.00000	0.2	0.8

Figure 1.14 Centrality for Induced Subgraph 1

node	centr_ degree_ out	centr_ eigen_ unwt	centr_ close_ unwt	centr_ between_ unwt	centr_ influence1_ unwt	centr_ influence2_ unwt
F	3	1.00000	1.00	0.16667	0.75	1.75
G	3	1.00000	1.00	0.16667	0.75	1.75
H	2	0.78078	0.75	0.00000	0.50	1.50
I	2	0.78078	0.75	0.00000	0.50	1.50

Figure 1.15 Centrality for Induced Subgraph 2

node	centr_ degree_ out	centr_ eigen_ unwt	centr_ close_ unwt	centr_ between_ unwt	centr_ influence1_ unwt	centr_ influence2_ unwt
J	2	1	1	0	0.66667	1.33333
K	2	1	1	0	0.66667	1.33333
L	2	1	1	0	0.66667	1.33333

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.

```
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 = LinkSetIn
   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

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

References

Raghavan, U. N., Albert, R., and Kumara, S. (2007), "Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks," *Physical Review E*, 76, 36106–36117.

Traag, V. A., Van Dooren, P., and Nesterov, Y. (2011), "Narrow Scope for Resolution-Limit-Free Community Detection," *Physical Review E*, 84, 016114 (1–9).
URL http://pre.aps.org/abstract/PRE/v84/i1/e016114