UCINET 6 for Windows
Software for Social Network Analysis

USER'S GUIDE

Borgatti, Everett and Freeman

2002

Last revised: 23 February, 2011

UCINET 6

Analytic Technologies
11 Ohlin Lane
Harvard, MA 01451 USA
Voice: (978) 456-7372
Fax: (978) 456-7373
Email: support@analytictech.com; sales@analytictech.com

Table of Contents

~ 0 ~
Preface

0.1 Notational Conventions

UCINET is menu-driven Windows program. This means you choose what you want to do by selecting items from a menu. Menus may be nested, so that choosing an item from a menu may call up a submenu with additional choices, which in turn may call up submenus of their own. Consequently, to get to certain choices, you may have to select through a number of menus along the way. To represent the options you must take to a given choice, we use angle brackets. For example, to run the hierarchical clustering procedure, you must first start **UCINET**, then click on the top toolbar and point to **Tools**, from the drop down menu that appears highlight **Cluster** and from the submenu that appears click on **Hierarchical**. We will represent this series of choices as

Tools>Cluster>Hierarchical

0.2 Acknowledgements

Dozens of people have contributed to UCINET 6 for Windows by making suggestions, contributing technical expertise, finding bugs, and providing moral and financial support. We especially thank Charles Kadushin, David Krackhardt, Ron Rice and Lee Sailer for their early and continued support. We also thank those who have contributed technical expertise for specific algorithms, including Pip Pattison (semigroups), Kim Romney (correspondence analysis) and Stan Wasserman (p1).

We are also grateful to Brian Kneller (University of Greenwich, London) for doing much of the initial programming on the Windows interface. The non-metric MDS program is adapted from UCINET 3.0 (MacEvoy and Freeman, 1985), which in turn was drawn from the University of Edinburgh's MINISSA program. Many of the procedures make use of routines found in *Numerical Recipes in Pascal* by Press, Flannery, Teukolsky and Vetterling, and *EISPACK* (Smith et. al. 1976, Springer-Verlag).

0.3 Programming Considerations

To paraphrase an old song (and reverse the meaning), UCINET 6.0 is built for speed, not for comfort. Oftentimes during the programming of UCINET, we had to choose between using a fast algorithm that used a lot of memory (and therefore reduced the maximum size of network it could handle), and a slow algorithm that saved memory and could handle much larger datasets. In previous versions we tried to strike a balance between the two. In this version, we usually chose speed. One reason for this is that it is precisely when working with large datasets that speed is essential: what good is an algorithm that can handle thousands of nodes if it takes days to execute? The other reason is that advances in hardware and operating system software continually extend the amount of memory programs can access, so it seems a waste of programming time to work out ways to economize on memory.

One consequence of menu systems is the need to organize program capabilities into categories and subcategories in a way that is logical and comprehensible. Of course, this has proved to be impossible. With only a little effort one can discover several competing schemes for classifying all the functions that UCINET 6.0 offers. None of the schemes is perfect: each does an elegant job of classifying certain items but seems forced and arbitrary with others. The scheme we have settled on is no exception. The basic idea is that under "network" we put techniques whose reason for being is fundamentally network-theoretic: techniques whose interpretation is forced when applied to

non-network data. An example of such a technique is a centrality measure. In contrast, under "tools" we put techniques that are frequently used by network analysts, but are also commonly used in contexts having nothing to do with networks. Multidimensional scaling and cluster analysis are examples of such procedures. Inevitably, of course, there are techniques that are either difficult to classify or for some reason are convenient to misclassify.

0.4 Content

UCINET 6.0 is basically a re-engineered version of UCINET IV, and so users familiar with UCINET IV should easily adapt to the new environment. We have extended UCINET's capabilities and re-organised the routines into what we believe to be more sensible categories.

Perhaps the most fundamental design consideration we have faced is choosing what capabilities the program should have. Since Freeman's first version was released, UCINET has incorporated a diverse collection of network techniques. The techniques are diverse both in the sense of what they do (detect cohesive subgroups, measure centrality, etc), and where they come from (having been developed by different individuals from different mathematical, methodological, and substantive points of view). In UCINET 6.0 we continue that tradition, seeing ourselves more as editors and publishers of diverse works than as authors with a single pervading perspective.

One problem with this approach is that different techniques implicitly assume different views of what their data are. For example, graph-theoretic techniques describe their data in terms of abstract collections of points and lines, algebraic techniques view their data as sets and relations, and statistical techniques understand their data to be variables, vectors, or matrices. Sometimes the mathematical traditions intersect and the same operation is found in the repertoire of each tradition; cognates, if you will. For example, a graph automorphism, which is a 1-1 mapping of a graph to itself, corresponds in the world of 1-mode matrices to a re-ordering of the rows and corresponding columns of a matrix. Likewise, the converse of a graph, in which the direction of all directed arcs is reversed, translates to the transpose of a matrix, which is an exchange of rows and columns. Unfortunately, there are also false cognates among the lexicons. For example, a relation may be expressed as a matrix, but in discrete algebra, the "inverse" of a relation is the transpose of that matrix, not the matrix which, when multiplied by the original, yields the identity.

0.5 Matrix Orientation

In UCINET 6.0, all data are described as matrices. While the prompts and outputs of some procedures may reflect the language most commonly associated with that technique (e.g. "networks" for centrality measures, "relations" for ego-algebras), it is extremely important for the user to maintain a "matrix-centered" view of the data. All UCINET data are ultimately stored and described as collections of matrices. Understanding how graphs, networks, relations, hypergraphs, and all the other entities of network analysis are represented as matrices is essential to efficient, trouble-free usage of the system.

~ 1 ~
Getting Started

1.1 Hardware

UCINET 6.0 requires a computer running Windows 95 (from 1997 or more recent), Windows 98, NT or other compatible operating system. The program requires 2mb of hard disk storage space and 16mb of RAM.

1.2 Installation

The UCINET program must be installed before it can be used. If you have a CD then place the disk in the drive and the program should start the installation automatically. If this does not happen then from the **start** button select the **run** option change to your CD drive and click on "setup.exe". If you have an electronic version then from **run** select the folder containing UCINET and select the file "setup.exe". The installation wizard will guide you through the installation procedure.

1.3 Quitting the Program

To leave the program, choose F**ile>Exit** from the toolbar, click on the door icon, or press **Ctl**+**x** (the **Control** key and the **x** key together). If the program is in the middle of executing an analysis and you want to interrupt it, you can try pressing **esc**. This works only for iterative procedures such as MDS or TABU SEARCH. Otherwise press **Control Alt Delete** (simultaneously) and this brings up the operating system window click on the button marked task manager, you can now select UCINET and end the task. Some procedures offer a "Calculating …" dialogue box with a STOP button that will stop execution at the next convenient point.

1.4 Technical Support

Registered owners of UCINET 6.0 (professional version) are entitled to technical support. Please e-mail, write or call the authors during standard business hours with any questions, suggestions and bug reports. They can be reached at the following addresses:

Steve Borgatti
Analytic Technologies
11 Ohlin Lane
Harvard, MA 01451 USA
Tel: +1 978 456 7372
Fax: +1 978 456 7373
Email: support@analytictech.com

Martin Everett
School of Computing and Mathematical Sciences
University of Greenwich

30 Park Row
Greenwich
London SE10 9LS
Tel: +44(0)20 8331 8716
Fax: +44(0)20 8331 8665
email: M.G.Everett@greenwich.ac.uk

Please communicate bugs as soon as possible: if at all possible, we will immediately replace the software with a corrected copy at our expense.

1.5 Citing The Program

If you use UCINET 6.0 to perform any analyses which result in a publication or presentation, you must cite it (this is required by your licensing agreement). The citation should treat the program as though it were a book. For example:

> Borgatti, S.P., M.G. Everett, and L.C. Freeman. 1999. *UCINET 6.0 Version 1.00*. Natick: Analytic Technologies.

In other words, the title of the "book" is the name of the program, and the publisher is **Analytic Technologies**. Since the program will change over time, it is important to include the version number in the title in order to allow others to replicate your work.

1.6 License and Limited Warranty

This software is protected by both United States copyright law and international treaty provisions. You must treat this software like a book, except that (i) you may copy it to the hard disk of a single computer or to the hard disk of any computer(s) that you personally have exclusive use of, and (ii) you can make archival copies of the software for the sole purpose of backing it up to protect your investment from loss.

By "treat this software like a book" we mean that the software may be used by any number of people, and may be freely moved from one computer to another, as long as there is no possibility of it being used simultaneously on two different computers. Just as a given copy of a book cannot be read by two different people in different places at the same time, neither can the software be legally used by different people on different computers at the same time.

Analytic Technologies warrants the physical CD and physical documentation associated with the UCINET 6.0 product to be free of defects in materials and workmanship for a period of sixty days from the date of purchase or licensing. If Analytic Technologies receives notification within the warranty period of defects in materials or workmanship, and such notification is determined by Analytic Technologies to be correct, we will replace the defective materials at no charge. However, do not return any product until you have called us and obtained authorization.

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of CD(s) or documentation and shall NOT include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit or prestige or data or use of the software, or special, incidental or consequential damages or other similar claims, even if Analytic Technologies has been specifically advised of the possibility of such damages. In no event will Analytic Technologies's liability for any damages to you or any other person ever exceed the price of the license to use the software, regardless of any form of the claim.

Analytic Technologies specifically disclaims all other warranties, express or implied, including but not limited to, any implied warranty of merchantability or fitness for a particular purpose. Specifically, Analytic Technologies makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty-day duration of the Limited Warranty covering the physical CD(s) and physical documentation only (and not the software) and is otherwise expressly and specifically disclaimed.

This limited warranty gives you specific legal rights; you may have others which vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

This License and Limited Warranty shall be construed, interpreted and governed by the laws of Massachusetts, and any action hereunder shall be brought only in Massachusetts. If any provision is found void, invalid or unenforceable it will not affect the validity of the balance of this License and Limited Warranty which shall remain valid and enforceable according to its terms. If any remedy hereunder is determined to have failed of its essential purpose, all limitations of liability and exclusion of damages set forth herein shall remain in full force and effect. This License and Limited Warranty may only be modified in writing signed by you and a specifically authorized representative of Analytic Technologies. All use, duplication or disclosure by the U.S. Government of the computer software and documentation in this package shall be subject to the restricted rights under DFARS 52.227-7013 applicable to commercial computer software.

~ 2 ~
The UCINET Environment

In this chapter we give an overview of running the program, covering such topics as menus and forms, the nature of UCINET 6.0 datasets, running analyses, and dealing with output.

2.1 Menus and Help

The program is menu-driven, which makes it very easy to use, even if you haven't used it in a while. After you start up the program, you will find yourself in the main window. The main window provides the following choices: **File, Data, Transform, Tools, Network, Options, and Help** together with three buttons. Each of these choices is itself a submenu with additional choices. The leftmost button directly calls the spreadsheet editor, as already stated the middle button exits from the program and the button on the right allows the user to change the default folder.

There are two ways to choose an item from a menu. The simplest method is to use the mouse to highlight the desired choice, then give a left click to activate a routine. If a highlighted choice has a submenu, that submenu will automatically appear. Another method is to press the highlighted letter of the desired choice, which is usually the first letter. For example, if the **Network** menu is highlighted then to get to **Roles and Positions** you can press **L**, you can now select another option if your choice is at the bottom level of the menu then the routine is activated. After selecting **Roles and Positions** typing **S** will select **Structural** and then **P** will select and run the **Profile** routine.

Sometimes, however, there is a way to circumvent the menus. Some menu items, such as displaying datasets found in **Data>Display**, have been assigned "hot-keys" which can be used to invoke that item directly without going through the menus. Provided no menus are highlighted then pressing **Ctrl+D** will immediately invoke the **Data>Display** routine.

All the hot keys are Control or Alt letter combinations. They are as follows:

Ctrl+F	**File>Change Default Folder**
Alt+X	**File>Exit**
Ctrl+S	**Data>Spreasheet**
Ctrl+D	**Data>Display**
Ctrl+B	**Data>Describe**
Ctrl+X	**Data>Extract**
Ctrl+M	**Data>Random>Matrix**

2.2 Forms

Suppose you choose a procedure from the menu. With few exceptions, the next thing you are likely to see is a *parameter form*. A form is a collection of one or more fill-in-the-blank questions. For example, suppose you click on **networks>Subgroups>Cliques** from the **File** menu. The form you see will consist of the following questions:

Input dataset:	Camp92.##h
Minimum size:	3
Analyze pattern of overlaps?:	YES
Diagram Type	Tree Diagram
(Output) Clique indicator matrix:	CliqueSets
(Output) Co-membership matrix:	CliqueOverlap
(Output) Partition indicator matrix:	CliquePart

The left-hand items (in **bold** type) are the question. The right-hand side contains the default answers. Often you will leave the default values as they are, expect for the input dataset, which is where you enter the name of the dataset that you want to analyze. You can either type the name yourself or double-click on that area and a dialog box will appear allowing you to choose the file you want from a list.

Once all the form is completed to your satisfaction, you activate the routine by clicking on the OK button (not shown above).

The next time you choose **Clique** from the menu, you will find that the default input dataset is whatever dataset you used the last time you ran **Clique**, unless the option **Smartdefaultnames** is on. In that case, the program puts in the default name that it thinks you might want to use, based on what you have done earlier. If this is the file you want to edit, just click on **OK**. If not, just start typing the name of the new file or click on the button to the right of the box. The moment you hit a key, the old filename will vanish: there is no need to backspace to the beginning of the name. If the filename is almost right but needs editing, use the mouse to place the cursor in the correct position so that you can fix it.

Whenever you are being prompted for a filename, then the button to the right of the box with three dots on it will activate the standard Windows procedure for selecting a file.

Incidentally, whenever you are called upon to enter a list of actors, you should refer to the actors by number (i.e. row or column in the data matrix) rather than by name or other label. Separate the numbers by spaces or commas. UCINET gives help in this process by allowing you to click on labels and automatically entering in the correct actor or matrix numbers. The button labeled with an **L** immediately next to the file selection button will give a list of labels you can now select from this list using the mouse. To select more than one entry you should press the **Control** key at the same time as you click on your choice. In addition, you can indicate groups of numbers using the following conventions:

3 to 10
3-10
first 5
last 30

The first two lines both specify the set of id numbers 3, 4, 5, 6, 7, 8, 9, 10. The last line only works when it is clear to the program what the total number of actors is. All of these conventions can be mixed, as follows:

List of actors to DROP: first 4, 18 12,5 19 to 25, last 2

2.3 Running An Analysis

As the diagram below indicates, the typical UCINET 6.0 procedure takes two kinds of input, and produces two kinds of output.

One kind of input consists of UCINET 6.0 datasets. Most procedures take a single dataset as input, but that dataset may contain multiple matrices, often representing different social relations measured on the same actors. Other procedures (such as the **Join** and **QAP Regression** procedures) take multiple datasets as input, each of which may contain several matrices.

Another kind of input consists of a set of parameters which either describe the data or alter the way the program runs. For example, as you saw earlier, the parameters for the **Clique** procedure are: the name of the input dataset, the minimum size of clique to report, and the names of various output files. Most parameters have default values which the user will have an opportunity to either accept or override.

Parameters are specified in forms that appear immediately after a procedure is selected from the menu. The first time a procedure is invoked in a given session, the fields for all parameters will contain "factory-set" default values (wherever possible). If you change the values of any parameters, these changes will remain in effect for subsequent runs, until you change them again or exit UCINET. The only exception to this rule occurs when some default settings depend on the values of other default settings, in which case changing certain parameter values will cause the program to change other defaults, even if you have previously set them.

One set of parameters that always have default values is the names of a procedure's output files. Output files are UCINET 6.0 datasets that can be read by all of the other UCINET procedures and therefore used as input for further analyses. For example, one of the outputs of the CLIQUE program is an actor-by-actor matrix whose ijth cell gives the number of cliques that actors *i* and *j* are both members of. This matrix is suitable for input to the MDS program to obtain a spatial representation of the pattern of overlaps. By default, the name of this dataset is **CliqueOverlap**.

In addition to output datasets, most UCINET procedures also produce a textual report as output. This report is always saved to an text file called a **Log File** (usually stored in the \Windows\System directory), and is also automatically displayed on the screen.

Let us run through an example of a clique analysis. The first step is to clique on **Network>Subgroups>Cliques** from the toolbar. This will pop up a form that contains the following:

```
Input dataset:
Minimum size:                          3
Analyze pattern of overlaps?:          YES
Diagram Type                           Tree Diagram
(Output) Clique indicator matrix:      CliquesSets
(Output) Co-membership matrix:         CliquesOver
(Output) Partition indicator matrix:   CliquesPart
```

The first line asks for the name of the UCINET 6.0 dataset containing the data. If the dataset is called **TARO** and is located in the **\uci3** folder of the **c:** drive, you would fill in the blank with **c:\uci3\TARO**. (Of course, naming the drive and folder is only necessary if the data are located in a different drive/folder than the current default). You could also select this file by clicking on the button to the right of the question and selecting the file using the mouse from the window that opens up.

The second line asks for the minimum size of clique to report. A default answer of 3 is already filled in. To override the default, just type over it.

The third line asks whether to compute and analyze an actor-by-actor matrix that counts up, for each pair of actors, the number of cliques they belong to in common. This matrix is then submitted to hierarchical clustering. The default answer here is *YES*, but to save processing time you may wish to override the default when working with large datasets.

The fourth line asks a question about what type of clustering diagram you would like to view. The default is a tree diagram; the only other alternative is a dendrogram which is a variation on the same theme.

The fifth through seventh lines ask for the names of datasets to contain the key outputs from the analysis. These datasets can then be used as input to other analyses. Default names are supplied for all three.

When you are through entering or modifying these parameters, click on **OK** to begin the analysis. Since TARO is a standard UCINET dataset then you will be able to run this analysis. The result will be a tree diagram appearing on the screen. If you now clique on the OK button then the following Log File will be displayed.

```
CLIQUES
------------------------------------------------------------------------------

Minimum Set Size:  3

Input dataset:     C:\Uci3\TARO

10 cliques found.

   1:  2 3 17
   2:  1 2 17
   3:  17 18 22
   4:  4 5 6
   5:  4 6 7
   6:  5 20 21
   7:  8 9 10
   8:  11 20 21
   9:  12 13 14
  10:  12 14 15


Group Co-Membership Matrix

                        1 1 1 1 1 1 1 1 1 2 2 2
     1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
     - - - - - - - - - - - - - - - - - - - - - -
  1  1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  2  1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0
  3  0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
  4  0 0 0 2 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  5  0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
  6  0 0 0 2 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  7  0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  8  0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
  9  0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 10  0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 11  0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0
 12  0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 0 0 0 0 0 0 0
 13  0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
 14  0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 0 0 0 0 0 0 0
 15  0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0
 16  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
 17  1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 1 0 0 0 1
 18  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1
 19  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 20  0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 2 2 0
 21  0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 2 2 0
 22  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1

SINGLE-LINK HIERARCHICAL CLUSTERING

            1 1 1 1 1 1 1         1 2 2       1 1 2
Level   8 9 0 3 2 4 5 6 9 5 6 4 7 1 1 0 1 3 2 7 8 2
-----   - - - - - - - - - - - - - - - - - - - - - -
    2   . . . . XXX . . . . XXX . . XXX . . XXX . .
    1   XXXXX XXXXXXX . . XXXXXXXXXXXXX XXXXXXXXXXX
    0   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX



Group indicator matrix saved as dataset CliquesSets
Clique co-membership matrix saved as dataset CliquesOver
Clique co-membership partition-by-actor indicator matrix saved as dataset CliquesPart

Date and time: 15 Jan 99 13:38:43
Running time:  00:00:01
```

The output from the **CLIQUE** program is typical of most of the analytical procedures. It begins with a report of the parameter settings used to make the run. This makes it easy to interpret and reproduce the output at a later time.
The next bit of output is a report on the number of cliques found, followed by a listing of each clique (one to a line). The first number on each line identifies the clique. The remaining numbers identify the actors that belong in that clique. If the data had contained actor labels (names), you would see names instead.

After the listing of cliques is an actor-by-actor clique co-membership matrix that gives the number of cliques each pair of actors has in common. This is the basis for the next table, which gives the results of a single-link hierarchical clustering of the co-membership matrix. If you would prefer a different clustering algorithm, you are free to submit the co-membership matrix, which is automatically saved , to the clustering program of your choice.

The final bit of output is a set of statements indicating what output files were created by the procedure. The main output file is a clique-by-actor binary matrix that indicates which actors belong to which clique. This matrix can be analyzed further, or used to extract a subgraph from the larger network using the **Data>Extract** procedure. For example, if you want to pull out just the network of relationships among members of, say, the fourth clique, just tell the **Extract** program what the name of the original dataset is, what the name of the clique-by-actor indicator matrix is (the default is CliquesSets) and which clique you want (#4 specified as ROW 4). That's all there is to it.

2.4 The Log File

Textual output such as shown above is normally written to a text file in the \windows\system directory whose default name is **Log File Number x**, where x is an integer. When you run an analysis, the program writes output to the Log File and then displays the contents to the screen the number of the file is displayed in a box in the tool bar. The file can edited, saved, printed in the normal way using the buttons and options on the toolbar. When you run another procedure a new Log File with a number one higher than the last is created. You may leave Log Files open on the screen or you may close them. You can re-open a previously closed Log Files by clicking on **File>Edit Previous Log File,** this opens up a window which displays the number of the last Log File and gives a list of the existing Log Files together with what routine was run to create them. Simply select a file or type in its number to

open up a closed file. As a default UCINET will keep 50 previous files but this can be increased or decreased by selecting **Options>Number of Log Files** and changing the value. If you wish to keep a Log File from one session to the next then it must be saved, this can be done by clicking the save icon on the procedure output window or selecting **Save** from the **File** option again in the procedure output window. Note that any unsaved log files are deleted as soon as you exit from UCINET.

2.5 Datasets

There are three important things to keep in mind about UCINET 6.0 datasets. The first is that they are collections of one or more matrices. It doesn't matter whether you think about your data as a graph (i.e., a set of vertices and a set of edges or links), a relation (i.e., a set of ordered pairs), a hypergraph (i.e., a set of subsets), or anything else: as far as UCINET is concerned, your data are a collection of matrices. This does not mean that UCINET cannot read data that are not in matrix form: it can (see Chapter 3 "Importing Data" in this Guide). It just means that once the data are in the system, it is thought of as a matrix.

Network analysts commonly think of their data as *graphs*. A graph is a set of points (also known as nodes or vertices) together with a set of lines (links, ties, edges) that connect the points. The information in a graph (who is connected to whom) can be represented by a matrix known as the *adjacency matrix*, in which a given cell X(i,j) contains a value of 1 if nodes *i* and *j* are connected, and 0 otherwise. In more precise language, a graph G = (V,E) with vertex set V and edge set E can be represented as a square symmetric 1-mode matrix *X*, known as the *adjacency matrix*, in which X(i,j) = X(j,i) = 1 if (i,j) belongs to E and X(i,j) = X(j,i) = 0 otherwise. Thus, the rows and columns of the adjacency matrix correspond to the nodes of the graph, and the cells in the matrix correspond to pairs of nodes or *dyads*. A matrix value X(i,j) = 1 indicates the presence of a link between node *i* and node *j*, and X(i,j) = 0 indicates the absence of a link.

Here is an example of a matrix representing a network:

	A	B	C	D	E
A	0	1	0	1	1
B	1	0	1	0	0
C	0	1	0	0	1
D	1	0	0	0	0
E	1	0	1	0	0

In this network, actor A has a tie with actors B , D and E, but not with C and not with him/her self. Actor B has a tie with A and with C, actor C has a tie with B and E, actor D has a tie only with A, and actor E has a tie with A and C.

A *directed* graph is a set of points and a set of *arcs* (also known as arrows or lines with heads and tails) that connect them. They are used to represent relations among nodes which are not necessarily reciprocal, as in "is in love with" or "is the boss of". The information in a directed graph can be recorded as a square 1-mode adjacency matrix (not necessarily symmetric) where X(i,j) = 1 if i is connected to *j* and X(i,j) = 0 otherwise. Note X(i,j) may equal X(j,i), but is not required to.

A *valued* graph is represented by a square, 1-mode matrix in which X(i,j) gives the value of the link from *i* to *j*, which might represent a strength of relationship, a length of a road, a probability of a state transition, a frequency of interaction, etc.

A *hypergraph* is a collection of subsets of a set of nodes. The subsets are conceptually like edges/links that may have more than two endpoints. In matrix form, a hypergraph is represented by a 2-mode *incidence* matrix in which Y(i,j) = 1 if node *i* is contained in subset *j*, and Y(i,j) = 0 otherwise.

The matrices contained in a UCINET dataset can have any shape or size, and they do not have to represent networks. For example, the following three collections of numbers are all matrices:

Matrix #1:

1 3 2 5
1 5 7 2
1 2 7 2
2 4 5 2
9 6 5 1

Matrix #2:

1 3 8 9 2 3 5 1.7

Matrix #3:

3.1415

Note that the second matrix has 8 columns and 1 row. The third matrix has 1 row and 1 column. Odd shapes are not a problem. All that is important is that every row contains the same number of columns, and vice-versa.

A useful feature of UCINET datasets is that they may contain more than one matrix, though the rows and columns of each matrix must correspond to the same objects. This enables you to place in a single file all network data relating to a single set of people. For example, you might have a set of families as nodes, and measure two relations on these families: "is married to a member of" and "has done business with a member of". This is useful for applying network techniques that take as input one or more social relations, such as most of the positional methods (e.g., CONCOR, REGE). Placing multiple relations in a dataset can even be useful when using techniques that do not apply to multiple relations, such as centrality measures. In UCINET, wherever possible, a procedure implementing a technique that does not make sense for multiple relations will run the technique on each relation sequentially. One application of this is to run, say, a centrality measure on a dataset containing several hundred random networks. The program will compute and save its measures for each network in the file. The results can then be analyzed statistically.

Another way of using multi-matrix datasets is implemented in the **Tools>Matrix Algebra** procedure. Here, the program "thinks" of a multi-matrix dataset as a single 3-way matrix composed of rows, columns and levels, and allows the user to perform operations on all three dimensions.

The second important thing to understand about UCINET datasets is that they are not text files. Therefore you cannot use a word processor or editor to enter or change them. Only UCINET (and other software by Analytic Technologies) can read and write them. This can be inconvenient, but it is well worth the significant improvement in performance that it buys. Of course, UCINET also provides a way to convert text files (and Excel spreadsheet files) into UCINET datasets (see the **Import** command) and vice-versa (see the **Export** command). In this respect, UCINET is similar to SYSTAT, SAS, SPSS, GRADAP and other well-known analytic packages.

The third important thing is that a single UCINET dataset actually consists of two physical files. One (with extension **.##**D) contains the actual data, and the other (with extension **.##**H) contains information about the data. In referring to an UCINET dataset, however, you will only refer to the ##H file (or just skip the extension altogether). Instead, you will use the filename proper, or filename.##H, as in "sampson" or "sampson.##h". Filenames can contain spaces and can start with a numeric character. However, occasionally, you will need to enclose a filename that contains spaces within quotation marks. UCINET IV datasets are compatible with UCINET 6.0 and no conversion is necessary.

2.6 The Default Directory

In any form, if you enter the name of file to be analyzed (e.g., **camp92**), UCINET assumes the file is in the current directory, unless you enter a full pathname (e.g., **c:\program files\UCINET 6\datafiles\camp92**). To change the default directory, click on the button on the far right of the main UCINET form that looks like a file cabinet. This will open a tree-like dialogue form that will allow you to choose the folder that you want to read and write datafiles to and from. Make sure to double-click the direct

2.6 Program Organization

UCINET is a diverse collection of hundreds of procedures and techniques drawn from a variety of sources. We have organized these myriad capabilities into six basic categories, corresponding to the following six items in the main window: **File, Data, Transform, Tools, Network and Options.**. In this section, we describe what kinds of routines are to be found under each heading.

2.7 File Submenu

The **File** submenu comprises routines which deal with files and folders together with the print set up and a command to exit UCINET. You can change the default folder, this is the folder where UCINET will look for a file if you do not provide it with a full path name. You may also create folders in which to keep some or all of your files. There are routines to **rename, copy** and **delete** UCINET files, useful because UCINET datasets consist of two physical files, which means that operations like deleting would otherwise have to be performed twice. There are also routines to **edit** text files and review and to **review** or **edit Log Files**.

2.8 Data Submenu

The **Data** submenu contains routines for managing UCINET datasets. It is divided into six basic sections. The first section contains just one routine the spreadsheet editor for inputting and editing UCINET files directly. The second section contains routines to create new UCINET datasets or bring other forms of data into UCINET. These include **Random** to create random data based on different distributions, **Import** for converting ASCII files of various types into UCINET datasets; and **Export**, for converting UCINET datasets into ASCII files that other programs can read together with three routines (**Attribute, Affilations** and **CSS**) for converting specific types of data into standard network data. The next section contains **Display**, which displays the contents of UCINET dataset in matrix form, and **Describe** which describes the meta-information available on UCINET dataset (such things as the number of rows and columns, labels and titles, etc) and also allows the user to input or edit labels. The fourth section consists of routines to **Extract** parts of a dataset to form a new dataset, **Join** together two or more datasets and **Unpack** a multirelational dataset into individual matrices. The next section of routines for performing non-numeric transformations and manipulations of UCINET datasets. They include: **Sort**, for sorting rows and columns of a matrix according to the values of a given variable; **Permute**, for re-ordering rows, columns and matrices into a specified order; and **Transpose** for interchanging the rows with the columns. The final section allows the user to re-organise the data in a variety of ways.

2.9 Transform Submenu

This submenu contains routines for transforming graphs and networks into other kinds. The submenu is divided into four sections. The first section contains routines that combine rows and/or columns. **Block** creates block densities which can then be converted into blockmodels, **Collapse** performs a similar but more general function allowing for rows and columns to be treated separately and giving the user options on how to calculate the combined values. The second section contains routines that typically operate on all cells of a matrix without changing its dimensions. Operations implemented include symmetrizing, dichotomizing, recoding, reversing and the **Diagonal** command, which enables the user to change the values of the main diagonal of a matrix, and to save the current values to a file. The third section contains routines **Rank** and **Normalize,** the first routine converts lists to ranks and the second normalizes either the whole matrix or just the rows (or columns) using a variety of techniques. The final section contains routines that usually result in either additional nodes, lines or additional relations. They include; **Linegraph**, which creates a graph in which the nodes correspond to the lines of an original graph; **Incidence**, which converts an adjacency matrix to a rectangular node-by-line indicator matrix; **Multigraph**, which converts a valued graph into a collection of binary adjacency matrices, one for each value in the graph, **Bipartite** which converts an incidence matrix of a bipartite graph into an adjacency matrix, **Multiplex** which constructs multiplex graph from a multirelational graph and **Semigroup** which generates the semigroup from generator matrices.

2.10 Tools Submenu

UCINET provides a number of tools that although they are not strictly network procedures have been used widely by network analysts. The submenu contains routines for metric and non-metric multidimensional scaling, cluster analysis, correspondence analysis, singular value decomposition, factor analysis and measures of similarity and dissimilarity. The **Statistics** section has standard descriptive methods the P1 model and a number of permutation test methods. These include **Matrix QAP** (regression and correlation), **Autocorrelation** as well as permutation versions of **Regression, Anova and T-Tests**. This section also includes some routines for constructing dendrograms, scatterplots and tree diagrams from UCINET datasets.

2.11 Network Submenu

Under **Cohesion** are found routines for evaluating distances among nodes, maximum flows between pairs of nodes, reachability, volume of paths between nodes, etc. **Regions** has methods for finding **Components** (strong and weak), **Bi-components** and **K-Cores**. Under **Subgroups** are found routines for finding various types of cohesive subsets proposed in the network literature, including cliques, n-cliques, n-clans, k-plexes, lambda sets and factions. **Ego Networks** calculates a variety of ego-based measures for every actor in the network. Under **Centrality** are found routines for computing various measures of node centrality, including degree, closeness, betweenness, flow betweenness, information centrality, eigenvector centrality, power and the measures of Katz and Hubbell. **Core/Periphery** leads to two routines for detecting core/periphery structures and locating each actor's position in the structure. Under **Roles and Positions** are found routines for clustering or classifying actors based on several definitions of structural similarity: structural equivalence, automorphic equivalence, and regular equivalence. Several algorithms are available for computing each kind of equivalence (e.g**. CONCOR** and profile euclidean distance for structural equivalence; **REGE** and **CATREGE** for regular equivalence; and **MAXSIM** and **ExCatRege** for exact equivalence). **Properties** has routines for computing density and transitivity of the network as a whole

2.12 Options Submenu

These are routines for changing default settings of UCINET 6.0 system parameters, such as data checking, display options and page size.

~ 3 ~
Importing Data

Before you can do any analysis using UCINET 6.0, you must create a UCINET dataset. Typically, network data "arrive" in the form of filled-out questionnaires or data sheets with entries taken from books and journals (as in data on trade between nations). In both cases, the medium of delivery is paper and so you must type the data into a file on the computer. The best, most general way to do this is to enter the data into an ASCII file using a file editor or your favorite word processor. The data may be entered in any of several different formats outlined in this chapter. Once the data are on disk, you can use the **Data>Import/Export>Import** command to convert these data into a UCINET 6.0 dataset. For users of statistical software like SYSTAT, this should be a familar step. In SYSTAT, you use the **DATA** command to read an ASCII file and create a SYSTAT system file, which the statistical procedures can then read. SPSS and SAS also have analogous procedures, although they do not require you to create a permanent system file as SYSTAT and UCINET 6.0 do.

Import can process ASCII data in a variety of file types. At the most general level, four kinds of data files are supported: Raw, DL, Excel and Ucinet 3 (UCINET 6.0 uses the same datasets as UCINET IV so these do not be imported). Raw files are those that contain only numbers, such as a respondent-by-variable matrix recording the numerically coded responses of a set of respondents to a set of questions. DL files contain the same data as Raw files, but in addition contain information about the data, such as the number of rows and columns, the names of variables, the name of the study, and so on. UCINET 3 files are similar to DL files, but more limited in the range of information provided. Excel are standard excel spreadsheet datafiles. Each of these file types is discussed in detail in the next sections.

If you are entering data from scratch, we **strongly** suggest that you use the DL filetype (you can always **Export** the data to another filetype at any time). This filetype is the most flexible in accepting data in a variety of formats.

Regardless of the type of file you import, the outcome is the same: a UCINET 6.0 dataset that can be used as input to any numerical procedure. However, you should be aware that you have a few choices about how your data are stored. UCINET can store data in one of three basic datatypes: Byte, Smallint, and Real. When you **Import** data from an ASCII file, you will be asked to select one of these three types although the best advice is to leave the default setting of real unless you have a very large dataset.

The Real datatype is the most powerful. Real datasets can contain values ranging from -1E36 to +1E36. They can also contain missing values, which are internally stored as the value 1E38. The disadvantage of the Real datatype is that 4 bytes of storage are required to represent each value, which can mean pretty hefty files. For example, a Real dataset containing two 150-by-150 matrices requires 176kb of disk space.

The Smallint datatype requires only 2 bytes of storage for each value, but can only represent whole numbers in the range -32000 to +32000. Missing values are not allowed.

The Byte datatype is the most economical, requiring only 1 byte of storage for each value. It is usually the best choice for binary data. Byte datasets can contain whole numbers ranging from 0 to 255, with no missing values.

If you need missing values, you must use Real datasets. However, you should be aware that most techniques of network analysis do not allow for missing values, and so only a few UCINET procedures know what to do with them. Those procedures that cannot handle missing values automatically transform them to zeros or other reasonable values.

In choosing a datatype, it is important to realize that while your choice affects the amount of disk space used to store the data, it has no effect on the amount of memory the program needs to process the data. A program such as **MDS** which is built to handle real values will read the data as real no matter how they are stored on disk. Similarly, a program that can only handle integer data, such as **Clique,** will automatically round real data as they are read into memory.

A note about missing values: All data values greater than 1E37 (10 raised to the power 37) are considered missing. Also any non-numeric character found in an ASCII datafile (when a number is expected) is considered missing as well. For example, the following matrix contains three missing values:

0	9.7	1E38	.
16.	0	3.1	na
18.1	1e9	0	-1.2
a3	12	.013	0

Note that unlike SYSTAT, UCINET 6.0 reads a lone period as zero, not as missing. If you are importing a SYSTAT dataset that contains missing values, you should use a text editor to change all lone periods to, say, 'NA'.

3.1 Raw Filetype

A raw data file is one that contains only numbers (no labels, titles or other information about the data), and which is typed in the form of a matrix. For example, a raw data file looks like this:

```
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
```

To determine how many columns there are in the matrix, the program reads the first line and counts the number of values. To determine how many rows there are, it just counts the number of lines in the file.

While this procedure is convenient, we do not recommend it. One reason is that there is no possibility of data checking by the computer: if the first row happens to be missing a number, the program will never know it, and will assume that the matrix has one less column than it really does. Another reason is that you cannot spread the data for a matrix row across more than one record in the data file. Furthermore, you can't use labels to identify nodes.

3.2 Excel Filetype

An Excel filetype is a data file created using Excel or saved as an Excel datatype from some other application package. Currently UCINET supports Excel versions 4.0, 5.0 (Office ’95) and 7.0 (Office ’97). If you are using other versions of Excel, then you must make sure that while in Excel you SAVE AS the data in one of the supported formats.

Note that Excel only supports up a maximum of 255 columns and therefore cannot be used to create large network datasets.

3.3 DL Filetype

A typical DL file consists of a set of numbers (the data) preceded by a series of keywords and phrases which describe the data and might therefore be described as "meta-data". Alternatively, a DL file can consist only of the meta-data, along with a pointer to another file that contains the actual data.

3.4 Full Matrix Format

A simple DL file for a 4-actor network might look like this:

```
dl n=4 format=fullmatrix
data:
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
```

The "dl" keyword identifies the file as a DL filetype, and must be the first word in the file. The phrase "n=4" indicates a matrix with 4 rows and columns. The equals sign may be replaced by one or more spaces or commas. All of the following are acceptable: "n = 4", "n 4", "n,4". The phrase "format=fullmatrix" indicates that the data are entered as an ordinary matrix (as opposed to a linkedlist, lowerhalf matrix, etc.). Since fullmatrix format is the default, this phrase is optional and may be dispensed with.

The "data:" keyword indicates that there is no more information about the data -- what follows are the data themselves. The order of the keywords is therefore important: if you had "dl data: n=4", the program would blow up. As we add other keywords to our repertoire, it is useful to keep in mind that the "dl" word goes first, followed by all phrases having to do with the dimensions of the matrix, followed by any other key phrases, followed by the "data:" keyword.

A note on punctuation: In general, a full colon (:) implies that a set of things are pending, such as a collection of data values or a collection of labels. A semicolon (;) or a carriage-return (i.e., a new line) imply the end of a phrase.

Each data value must be separated from the others by one or more spaces or carriage returns. All non-numeric values, except lone periods, are treated as missing values. The physical formatting of the data into rows with equal-length columns is not required, as long as all values are present and in left-to-right, top-to-bottom order. For example, the following is perfectly acceptable:

```
dl n=4 data:
0 1 1 0 1 0 1 1
1 1
```

```
0 0
0 1 0
0
```

3.5 Rectangular Matrices

Rectangular matrices might be entered as follows:

```
dl nr = 6, nc = 4
data:
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
1 0 1 1
1 1 0 0
```

The phrase "nr = 6" indicates that the matrix has 6 rows. The phrase "nc = 4" indicates that the matrix has 4 columns.

3.6 Labels

DL files may also contain actor labels, as follows:

```
dl n=4
labels:
Sanders,Skvoretz,S.Smith,T.Smith
data:
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
```

The "labels:" keyword indicates that the next 4 items are row and column labels. The labels may be up to 18 characters long (when the option **longlabels** is off) or 255 characters long (when **longlabels** is on), and are separated either by spaces, commas or carriage returns (or both). Labels cannot contain embedded spaces unless you enclose them in quotes, as in "Tom Smith".

The word "labels" should be followed by a full colon.

Labels may be specified separately for rows and columns. In fact, this is necessary when the matrix is not square. For example:

```
dl nr = 6, nc = 4
col labels:
hook,canyon,silence,rosencrantz
data:
0 1 1 0
1 0 1 1
1 1 0 0
```

```
0 1 0 0
1 0 1 1
1 1 0 0
```

Another way to enter labels is as part of the data matrix:

```
dl nr = 6, nc = 4
row labels embedded
col labels embedded
data:
      Dian Norm Coach  Sam
Mon     0    1     1    0
Tue     1    0     1    1
Wed     1    1     0    0
Thu     0    1     0    0
Fri     1    0     1    1
Sat     1    1     0    0
```

The "row labels embedded" and "column labels embedded" phrases indicate that row and column labels are embedded in the data themselves. It is also valid to write simply "labels embedded" to indicate that both row and column labels are found in the data.

3.7 Multiple Matrices

Sometimes it is convenient to have several, related, matrices in a single datafile. For example, we may measure several social relations among a given set of actors. Here is how to do it:

```
dl n = 4, nm = 2
labels:
GroupA,GroupB,GroupC,GroupD
matrix labels:
Marriage,Business
data:
0 1 0 1
1 0 0 0
0 0 1 0
1 0 0 1

0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
```

The phrase "nm=2" indicates that the file contains 2 matrices. The keywords "matrix labels:" indicates that the next two words ("marriage" and "business") are labels for each of the matrices, presumably indicating the social relation that each measures.

3.8 External Data File

Sometimes it is convenient to separate the data from the DL meta-data that describe them. This allows the data file to be read by other programs (like SYSTAT, STRUCTURE, and NEGOPY) as well as UCINET. Here is how it's done:

```
dl n = 16
labels:
ACCIAIUOL,ALBIZZI,BARBADORI,BISCHERI,CASTELLAN,GINORI
GUADAGNI,LAMBERTES,MEDICI,PAZZI,PERUZZI,PUCCI,RIDOLFI,
SALVIATI,STROZZI,TORNABUON
datafile C:\DATA\PADGM.DAT
```

The phrase "datafile = c:\data\padgm.dat" indicates the name of the file containing the actual data. That file should not contain anything but data.

The disadvantage of using the *datafile* approach is that it multiplies the number of files that must be kept track of.

When a separate datafile is used, UCINET checks the first line of the file to see if it contains a FORTRAN format statement such as NEGOPY and STRUCTURE require. If it does, UCINET begins reading the data from the second line. Otherwise, it begins at the top of the file. This allows you to use STRUCTURE and NEGOPY files interchangeably with UCINET files, as long as each value in the file is separated from the next by at least one space.

3.9 Diagonal Absent

By default, the program assumes that the data consist of a whole matrix with no cells omitted. Technically, this is referred to as the "fullmatrix format" and may be explicitly specified as follows:

```
dl n=4 format=fullmatrix data:
0 1 1 0 1 0 1 1
1 1
0 0
0 1 0
0
```

However, in the case of square matrices, it is sometimes convenient to omit some values. For example, it is possible to omit the diagonal:

```
dl n = 4
diagonal = absent
labels:
Sanders,Skvoretz
S.Smith,T.Smith
data:
  1 1 0
1   1 1
1 1   0
0 1 0
```

The program automatically fills in the absent values with missing value codes.

3.10 Lowerhalf and Upperhalf Matrices

Another possibility is to enter only the lower half of a square symmetric matrix:

```
dl n = 4
format = lowerhalf
diagonal = absent
labels:
Sanders,Skvoretz
S.Smith,T.Smith
data:
1
1 1
0 1 0
```

UCINET will automatically fill in the top half of the matrix to correspond to the bottom half, and fill the diagonal with missing values. It is also possible to enter an upper half matrix using the keyword "upperhalf" instead of "lowerhalf". Note that if the "diagonal absent" phrase is omitted, the program will expect a diagonal value to be present. For example:

```
dl n = 4
format = lowerhalf
labels:
Sanders,Skvoretz
S.Smith,T.Smith
data:
2
1 2
1 1 2
0 1 0 2
```

3.11 Blockmatrix Format

Another data format, useful for creating design or model matrices, is called "blockmatrix". In this format, rather than inputting the data directly, you input directions for creating the data. For example, to input a highly structured matrix like

```
2 1 1 1 1 0 0 0 0 0
1 2 1 1 1 0 0 0 0 0
1 1 2 1 1 0 0 0 0 0
1 1 1 2 1 0 0 0 0 0
1 1 1 1 2 0 0 0 0 0
0 0 0 0 0 2 1 1 1 1
0 0 0 0 0 1 2 1 1 1
0 0 0 0 0 1 1 2 1 1
0 0 0 0 0 1 1 1 2 1
0 0 0 0 0 1 1 1 1 2
```

using blockmatrix form, you would create the following input file:

```
dl n = 10 format = blockmatrix
```

```
data:
rows 1 to 10
cols 1 to 10
value = 0

rows 1 to 5
cols 1 to 5
value = 1

rows 5 6 7 8 9 10
cols 5 to 10
value = 1

diagonal 0
value = 2
```

In blockmatrix format, you identify a set of cells, then give them a value. In the example, the first three lines of data assign a value of 0 to all cells in the matrix. The next three lines (ignoring blanks), isolate the top left quadrant of the matrix and assigns all cells a value of 1. The next three lines do the same for the bottom right quadrant. The last two lines give a value of 2 to every cell along the main or 0th diagonal.

As another example, consider this matrix:

```
100   0   0   0   0   0
 90 100   0   0   0   0
 80  90 100   0   0   0
 70  80  90 100   0   0
 60  70  80  90 100   0
 50  60  70  80  90 100
```

To enter this matrix in blockmatrix format, enter the following lines:

```
dl n = 10 format = blockmatrix
data:
rows all
cols all
value 0
diag 0
val = 100
diag -1
val = 90
diag -2
val = 80
d -3
v = 70
d -4
v = 60
d -5
v = 50
```

Note the use of the keyword "all" to indicate all rows and columns. Note also that "rows", "cols", "value", and "diagonal" may be abbreviated to first letters.

3.12 Linked List Formats

An important set of formats for network analysis are the so-called linked list formats, in which the user specifies only ties that actually occur in the data and omits ties that did not occur. These formats are unique in being able to accept character data.

There are two basic types of linked list formats: "nodelists" and "edgelists". Each of these types in turn has two variants, one for 1-mode data and one for 2-mode data. Only the edgelists allow valued data.

3.12.1 Nodelists

A node list consists of a list of nodes that connected to a given node. Two nodelist formats are available: *nodelist1* and *nodelist2*. The first is used for square, 1-mode matrices such as networks or proximities. The second format is used for rectangular, 2-mode matrices such as a case-by-variable matrix or an item-by-use matrix. Both formats are able to accept character data. The *nodelist1* format looks like this:

```
dl n = 4, format = nodelist1
labels:
Sanders,Skvoretz,S.Smith,T.Smith
data:
1  2 3
2  1 3 4
3  1 2
4  2
```

The phrase "format=nodelist1" specifies a format in which the first number in every row of data identifies the actor (whom we'll call "ego") whose data the row pertains to. The remaining numbers on the row identify the actors that ego is directly connected to (i.e., his alters). For example, the third data line says that actor 3 (S.Smith) is connected to Sanders and Skvoretz. The order of egos (rows) is immaterial, as is the order of alters within a row. Egos without alters are simply left out (but still count in the "n = " phrase and still show up in the list of labels).

If you have too many alters to fit on one line, you can simply create a second line for that actor, but the first number in the new line must indicate who these alters belong to. For example, the following is equivalent to the file above:

```
dl n = 4, format = nodelist1
labels:
Sanders,Skvoretz,S.Smith,T.Smith
data:
 1  2 3
 2  1 3
 2  4
 3  1 2
 4  2
```

Notice how the ties for actor 2 are spread across two lines.

An alternative approach to entering the very same data is to use item labels instead of id numbers, as follows:

```
dl n = 4, format = nodelist1
labels embedded
data:
sanders   skvoretz s.smith
skvoretz  sanders s.smith t.smith
s.smith   sanders skvoretz
t.smith   skvoretz
```

It is possible to both list all the labels in the top half of the file (under a "labels:" statement as shown earlier), and have the labels embedded in the data (make sure the "labels embedded" command is also present). The advantage of this approach is that the rows and columns of the resulting matrix will be in the order you specify in the list of labels. For example, the following would create a matrix in which the rows and columns were sorted in alphabetical order:

```
Dl n = 4
format = nodelist1
labels:
Alfred, Betty, Chuck, David
Labels embedded
Data:
Betty chuck alfred
Alfred chuck david
David chuck betty
Chuck betty
```

Without listing the labels first, the resulting matrix would have been sorted in the order in which the labels were encountered: row/column 1 would have been Betty, row/column 2 would have been Chuck, row/column 3 would have been Alfred, etc.

The *nodelist2* format is used to input 2-mode data in which two different kinds of entities are being related: for example, persons and organizations, or persons and events. An example is as follows:

```
dl nr=3, nc=4 format = nodelist2
row labels embedded
column labels embedded
data:
benny sex_roles, stats
ginny aging, intro
sally stats, intro, aging
```

These data record which of three people teaches each of four courses. The first line of data indicates that Benny teaches Sex Roles and Statistics. The third line says that Sally also teaches Statistics, but teaches Intro and Aging as well. The first item in each line always refers to a row entity, and the rest of the items on a line refer to column entities.

3.12.2 Rankedlist Formats

The rankedlist formats are a simple variation on the nodelist formats that assume that the nodes attached to a given node have been entered in a meaningful order. The program then assigns tie strengths based on the ordering. For example, consider this file:

```
dl n = 4, format = rankedlist1
labels embedded
data:
sanders  skvoretz s.smith
skvoretz  sanders s.smith t.smith
s.smith  sanders skvoretz
t.smith  skvoretz
```

If we look at the first line after the Data statement ("`sanders  skvoretz s.smith`"), This says that for actor Sanders, the actor Skvoretz is ranked 1st, and the actor S.Smith is ranked 2nd. Actors not listed at all are assigned zeros by default, unless the phrase "recodena = no" appears in the dl language above the Data statement, in which case missing values are supplied instead. The result of reading the file above is the following matrix:

	Sanders	Skvoretz	S.Smith	T.Smith
Sanders	0	1	2	0
Skvoretz	1	0	2	3
S.Smith	1	2	0	0
T.Smith	0	1	0	0

Just like the case of nodelists, there is also a rankedlist2 that is used for 2-mode data:

```
dl nr=3, nc=4 format = nodelist2
row labels embedded
column labels embedded
data:
benny sex_roles, stats
ginny aging, intro
sally stats, intro, aging
```

In this case, the line "benny sex_roles, stats" would assign a 1 to sex_roles and a 2 to stats, for the benny row of the matrix.

3.12.3 Edgelist Formats

Instead of specifying a list of alters for each ego, the *edgelist* formats specify each link (or cell of a matrix) individually. Two edgelist formats are available: *edgelist1* and *edgelist2*. The first is used for square, 1-mode matrices such as networks or proximities. The second format is used for rectangular, 2-mode matrices such as a case-by-variable matrix or an item-by-use matrix. Both formats are able to accept character data.

The *edgelist1* format consists of a list of edges (i.e. links) followed optionally by their values, as shown in this example:

```
dl n = 4 format = edgelist1
labels:
Sanders,Skvoretz,S.Smith,T.Smith
data:
1 2 1
1 3 2
2 1 1
2 3 1
2 4
3 1 1
3 2 na
4 2 10.3
```

In the example, the fourth line of data says that actor 2 is connected to actor 3 with a tie-strength of 1. If the tie-strength is omitted, as in the fifth line of data, it is assumed to be 1.0. If a dyad is omitted, such as the (4,3) pair, it is assumed the strength of tie between them is 0.0. As with the matrix formats, tie strengths may be integers or real numbers, depending on the storage option selected. Non-numeric values, such as "na" are coded as missing. The adjacency matrix generated from this input data would look like this:

```
              Sanders   Skvoretz  S.Smith T.Smith

Sanders           0         1         2       0
Skvoretz          1         0         1       1
S.Smith           1                   0       0
T.Smith           0        10.3       0       0
```

Ties may be listed in any order, but only one tie may be specified per line.

One feature of the *edgelist1* format is the ability to cope with character data. This is principally useful when used in conjunction with a "labels embedded" statement as follows:

```
dl n = 3 format = edgelist1
labels embedded
data:
sanders skvoretz 1
sanders s_smith 1.782
skvoretz sanders 1.09
```

This causes the program to draw the labels for the rows and columns of the adjacency matrix from the first two fields in each data record, in the order in which they are found. Thus, in this example, a matrix with 3 rows and columns is specified where the first row/column is identified with "Sanders", the second is identified with "Skvoretz" and the third is identified with "S_Smith". The program will deal correctly with character identifiers of any length, but only the first 19 characters (or 255, if **longlabels** is on) will be stored as row/column labels.

Note that the rows and columns of the matrix formed from these data will, by default, be ordered alphabetically. For instance, in the data above, the first row will correspond to *s_smith*, the second to *sanders*, and the third to *skvoretz*. By adding the phrase “alpha = no”, you can prevent the automatic alphabetization, in which case the

rows and columns will be ordered according to the order in which they are encountered in the file. In the data above, the first row will correspond to *sanders*, the second to *skvoretz*, and the third to *s_smith.* You

You can have multiple relations in a single file by separating them with a vertical bar (|). For example:

```
dl nm=2 n = 3 format = edgelist1
labels embedded
data:
sanders skvoretz 1
sanders s_smith 1.782
skvoretz sanders 1.09
!
sanders s_smith 1
skvoretz sanders 1
```

The *edgelist2* format is identical to the *edgelist1* format, but is used with rectangular, 2-mode data. Consider, for example, the following matrix:

	COHESION	SIMILARITY	CENTRALITY
CLIQUE	1	0	1
CLOSENESS	1	0	1
REGE	0	1	0
DENSITY	1	0	1
CONCOR	1	1	0

An input file using the edgelist2 format is as follows:

```
dl nr = 5 nc = 3, format = edgelist2,
labels embedded
data:
clique cohesion
clique centrality
closeness cohesion
closeness centrality
rege similarity
density cohesion
density centrality
concor cohesion
concor similarity
```

In the example, all edges that are present have weight 1, hence no weights were specified in the data file. The following input file has weighted edges:

```
dl nr=2, nc=3 format = edgelist2
labels embedded
data:
sanders skvoretz 1
sanders s_smith 1.782
skvoretz sanders 1.09
```

While the data in this file are identical to the data used to illustrate the edgelist1 format, the results are very different:

```
           Skvoretz  S_Smith   Sanders
Sanders    1         1.782     0
Skvoretz   0         0         1.09
```

Note that only two different objects (Sanders and Skvoretz) occur as rows of the incidence matrix (see the first field of each data record), while three different objects (Skvoretz, S_Smith and Sanders) occur as columns of the matrix (see the second field of each data record). Note also that the fact that, in this example, the row objects happen to have the same names as some of the column objects means nothing: the program does not regard them as the same.

3.12.4 The EdgeArray1 Format

This format is a lot like an edgelist, but allows you to have multiple attributes of each tie (i.e., multiple relations). For example:

```
dl nm = 4 n = 3 format = edgearray1
labels embedded
matrix labels:
avgcalls
yearstogether
pctincommon
kin
data:
sanders skvoretz   1    62   33.7 0
sanders s_smith    1.782 29  66.2 1
skvoretz sanders   1.09 59   99.0 0
```

3.15 UCINET Spreadsheet Editor

You can type data in directly or cut and paste data from a table or spreadsheet into UCINET's spreadsheet. Simply start the UCINET editor by clicking on the editor button in UCINET or click on **Data>Edit** or type Ctrl+E . To copy your data from your spreadsheet or table simply move it onto the clipboard click on UCINET and paste the data onto the spreadsheet either by using Ctrl+V or by selecting **Edit>Paste** in UCINET. You then need to save your data within UCINET and it will be saved as a UCINET file. If you wish to type in your data directly then you may do so in the editor. Note that you can only type in the labels once some data has been filled in to the relevant row or column. If you already know the size of your data then fill in the last row and column entry first and you can type in the labels at the beginning. If your data is symmetric click the **Asymmetric** mode button before you enter any data this will automatically fill in the other half of your data. You need only enter the non-zero values in the spreadsheet, once these have been filled in then click on the button marked **Fill** all empty cells will be given a value of zero. If you accidentally stray outside the size of your required matrix then you need to delete the extra rows and columns rather than filling them in with blanks. If your data has more than one relation then add the extra matrices using the + button on the right side of the toolbar (the - can be used to delete relations). Individual matrices within the network can be named using the rename sheet button situated just to the right of the add and delete worksheet buttons.

The editor allows the access to some 2D and 3D graphics facilities. To utilize the graphics load a UCINET dataset into the editor. Block the data that you wish to display. Click on **edit>copy** to move the data onto the clipboard and then click on **edit>paste** to deposit the data into the spreadsheet graphics facility. Finally click on the graph button on the tool bar towards the right hand side just left of the **Symmetric/Asymmetric Mode** button. The graphic wizard will take you through the creation of your picture or chart.

Like Excel, the UCINET spreadsheet is limited to 255 columns and so this method cannot be used for larger datasets.

~ 4 ~
Data Processing

Having entered raw data into the system, you will often find it necessary to transform or modify them before submitting them to analysis. Sometimes, the reason for making changes is to correct errors or deficiencies in the data collection technique. For example, if you ask members of a group to rank-order each other in terms of "closeness", it may turn out that some of the lower rankings (say, below 10) were essentially random because the respondent didn't know who the person was. In such a case, you might decide to recode all ranks lower than 10 to a single value. Alternatively you may have simply mistyped one or two values, if the dataset is small then the easiest way to correct your error is to use the spreadsheet editor as outlined in the previous section. In this section we shall be concerned with more radical changes to the data.

A common example is when the data needs to be changed to allow their use in a certain analytic procedure which requires somewhat different data. For example, you might collect valued data on the strength of certain kinds of relationships among a set of people. If you then want to locate Luce and Perry cliques, you will need to dichotomize the data because cliques have only been defined for ordinary graphs in which nodes are either connected or they're not.

Another reason for transforming the data is to create a new variable, more useful analytically, that is derived from the collected data. For example, suppose you have attitudinal data on a set of actors for a particular set of issues. Suppose you also have relational data on the strength of friendships among actors. A natural hypothesis to check would be that similarity in attitudes is correlated with strength of relationship. You can test the hypothesis using the QAP procedure. First, however, you will have to obtain the similarity of attitudes among actors. This can be done by computing correlations among rows of the actors-by-attitudes matrix.

UCINET 6.0 can perform a wide variety of data transformations. Before describing various transformations, however, it is important to note that the use of the word "transformation" is misleading since none of the procedures actually change the existing dataset: rather, they generate a new dataset that reflects the appropriate changes. (Although this new dataset can be made to overwrite the existing one) It is this new dataset that is then submitted for analysis (or, more likely, further transformation).

4.1 Subgraphs and Submatrices

Perhaps the most common data adjustment is the removal of one or more nodes from a network, corresponding to removing one or more rows and columns from an adjacency matrix. The easiest way to do this in UCINET 6.0 is to run the **Data>Extract** procedure which extracts a submatrix from an input matrix. The inputs to this procedure are, first, a dataset to use as input, and second, a list of rows and/or coloumns to keep or delete. You have the option of specifying which ones to retain or which ones to drop. If you are just dropping one or two row or columns, it makes more sense to name those than to list all the rest. On running **Extract**, you will be asked to provide the following information:

```
Input dataset:
Are you going to KEEP or DELETE?  KEEP
```

```
|Which rows to KEEP?              ALL                    |
|Which columns to KEEP?           ALL                    |
|Which matrices to KEEP?          ALL                    |
|Output dataset:                  EXTRACT                |
```

Where it says, "Which rows to KEEP?" you enter a list of actors, such as "1 TO 10, 12 TO 24". You must do the same for the columns, as we are removing nodes from the network our columns to keep are the same as the rows, in this case you can just type "**same**". If the data has labels and you know the actor labels and not their number then click on the L button to the right of the box and you can select the appropriate actors from the list. When you do this **Ctrl** plus a mouse click will allow you to highlight multiple actors.

Alternatively, you can give the name of a UCINET 6.0 dataset containing a *group indicator matrix* such as output by several of the analytic procedures (e.g., **Cliques**). A group indicator matrix is a binary group-by-actor matrix in which X(i,j) = 1 if actor *j* is a member of group *i* and X(i,j) = 0 otherwise. For example, the **Cliques** program names the group indicator matrix it creates **CliquesSets**; if you would like to pull out a subgraph consisting only of the nodes in the sixth clique recorded in **CliquesSets**, fill in the form as follows:

Which rows to KEEP**? CliquesSets row 6**

Then either use "**same"** or type in "**CliqueSets row 6**" for the columns prompt. This says, 'The actors I want to keep are given by the 6th row of a UCINET 6.0 dataset called **CliquesSets**'. IMPORTANT NOTE: If you the dataset contains spaces inside the name, as in "**My Clique Sets**", you will need to put quotes around the filename:

Which rows to KEEP**? "My Cliques Sets" row 6**

Of course, the dataset you give does not have to have been created by the **Clique** program. You can create one yourself by simply creating an ASCII file containing a string of 1s and 0s and then **Importing** the information as a dataset. For example:

```
dl nr=1 nc=10 data:
1 0 0 1 1 1 0 0 1 0
```

If you import this data as dataset **mygroup**, you would then answer the prompt in **Extract** like this:

Which rows to KEEP? **mygroup row 1**

Again you need to repeat this at the columns prompt. This would cause a subgraph containing the relationship among actors 1, 4-6, and 9 to be created. Obviously there will be other cases when you will want to select different rows from the columns.

4.2 Merging Datasets

Sometimes it is convenient to keep different data collected on the same actors in different files. However, it can also be useful to join them together in a common file. For example, one may wish to perform several analyses on each of the Sampson relations separately, but for certain procedures (e.g., positional analysis) you may wish the analysis to be based on all measures simultaneously. The **Data>Join** procedure allows you to merge datasets with respect to three dimensions: rows, columns and levels (matrices). For example, consider the following two matrices:

Matrix number 1 is

```
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

Matrix number 2 is

```
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

To merge with respect to rows is to append the second matrix to the bottom of the first matrix to produce a new matrix with 8 rows and 4 columns, as follows:

```
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

To merge with respect to columns is to append the second matrix to the right side of the first matrix to produce a new matrix with 4 rows and 8 columns, as follows:

```
1 1 1 1 2 2 2 2
1 1 1 1 2 2 2 2
1 1 1 1 2 2 2 2
1 1 1 1 2 2 2 2
```

To merge with respect to levels is to place the two matrices in a dataset one after the other, to create a dataset with 2 matrices of 4 rows and 4 columns each, as follows:

```
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

4.3 Permutations and Sorts

Closely related to extracting subsets of matrices is the topic of re-ordering rows and/or columns. Often, reordering a matrix so that members of the same group or class are adjacent to each other makes it easier to grasp the overall

structure of relationships. UCINET 6.0 distinguishes between two basic types of reorderings: permutations and sorts.

Permutations are reorderings in which the user specifies the new order of nodes directly. For example, you might tell the **Permute** program that the ordering of nodes should be '1, 6, 2 to 5, 10 9 8 7', which means you want node 1 first, node 6 second, etc. For example, here is the input form for the **Data>Permute** program:

```
Input dataset:
New order of rows:
New order of cols:
New order of matrices:
Output dataset:                Permuted
```

In contrast to permutations, sorts are reorderings in which the new order is specified indirectly by reference to an attribute. For example, you might want the program to sort the nodes in order of decreasing centrality. To do this, you must first have a dataset on disk containing the centrality of each node. Then you tell the **Sort** procedure which dataset contains the criterion vector (i.e., the centrality vector), and which row or column of that dataset it is. Here is the input form for the **Data>Sort** program (listing default answers to questions):

```
Input dataset:
Dimensions to be arranged:      Both
Sort order:                     Ascending
Criterion vector (sort key):
Output dataset:                 Sorted
```

4.4 Transposing and Reshaping

It is often convenient to interchange the rows and columns of a matrix. For example, in the world system literature, it is commonplace to arrange network data such that the value of X(i,j) gives the dollar-amount of goods moving from country to *j* to country *i*. For networkers, this is confusing because in the network literature X(i,j) almost always indicates the value of an arc from *i* to *j*. All that is needed is to transpose the matrix using the **Data>Transpose** procedure. For another example, suppose you have a dataset with two rows of data and 50 columns. You would like a scatter plot of the first row against the second. However, the scatter plot program in UCINET can only plot columns. The solution is to transpose the data to give a matrix with 50 rows and 2 columns.

When you have 3-dimensional data (i.e., several matrices in a single dataset), the **Data>Transpose** procedure transposes each matrix in the dataset. However, what if you wanted to transpose not rows and columns, but columns and levels (i.e., matrices)? For example, the Newcomb year-1 dataset consists of 12 17-by-17 matrices, one for each week that data were collected. The rows refer to egos (respondents), the columns refer to alters and the levels refer to weeks. The values in the data refer to each ego's ranking of each alter on each week. Suppose that instead we wanted to organize the data so that each alter's ratings from a given respondent across weeks were side by side (so we could see whether they were increasing or decreasing in esteem). We would like columns of the matrix to refer to weeks, rows to refer to egos and levels to refer to alters. What we want, then, is to transpose (interchange) the columns and levels of the original data matrix. This is easily done using **Tools>Matrix Algebra**, as follows:

-->tnewc1 = transp(newcomb1 columns levels)

This says, 'interchange columns and levels of the dataset **Newcomb1** and place the result in dataset **Tnewc1**'.

A related procedure is the **Data>Reshape** facility. Unlike the **Transpose** procedure, this procedure changes the dimensions of a matrix without changing the order of data values. For example, suppose you have a 10-by-10 matrix. For certain purposes, it might be convenient to string out all the elements into a single 100-element vector. To do this in UCINET 6.0, run the **Reshape** program and tell it that you want the output to have 100 rows and 1 column. The data elements in the file will be written in the order in which they appear in the matrix, moving from left to right and top to bottom.

4.5 Recodes

One type of transformation that is frequently needed is to recode a range of values in a matrix to a new value. For example, it may be desirable to convert all values from 0 to 9 to 0, all values from 10 to 19 to 1, etc. The easiest way to do this is using the **Transform>Recode** procedure. Once you have invoked the procedure and filled out the first part of the form (specifying the input), you must fill out the following :

```
| Values     0      to      9      are recoded as    0      |
| Values    10      to     19      are recoded as    1      |
| Values    20      to    100      are recoded as    2      |
| Values    na      to     na      are recoded as    0      |
| Values            to             are recoded as           |
```

To fill in the form as we have, use the **tab** key or mouse to move from left to right and top to bottom. Note that on the fourth line of the form we have requested that missing values (na) be converted to zeros.

The **Recode** command is often useful for converting zeros to missing values and vice-versa. Note, however, that you cannot direct the program to change only values along the main diagonal, as might sometimes be needed. For this, you can use the **Transform>Diagonal** procedure. This procedure lets you change the values of the main diagonal of a matrix.

A special case of the general recoding problem is dichotomizing a matrix so that all values are either 0 or 1. Typically, this is needed when you have measured the strength of ties among actors, but for the purposes of a given analysis, would rather have binary data which simply record the presence or absence of a tie (or a strong tie). Sometimes this comes up because the UCINET procedure you want to run requires binary data. If so, the procedure will automatically change all values larger than zero to one. However, this may not be the ideal cutoff for your data (e.g., suppose your data are all negative numbers). By dichotomizing the data yourself ahead of time, you can ensure that they are dichotomized sensibly.

The **Transform>Dichotomize** procedure dichotomizes a matrix by comparing each cell value with a cutoff that you supply. Depending on your choice, the program can assign a 1 to any cell whose value is strictly greater than, greater than or equal to, exactly equal to, less than or equal to, or strictly less than the specified cutoff.

Another kind of recoding operation is the reversal of values (**Transform>Reverse**). For example, for a binary matrix with values 0 and 1, this would mean changing all 1s to 0s and vice-versa. This might be used to change a relation from "likes" to "does not like". For valued data, the procedure subtracts all values from the largest and adds the smallest. Hence, for data with values 0, 1, and 2, the procedure would change the 0s to 2s, the 2s to 0s, and leave the 1s as 1s.

4.6 Linear Transformations

Reversing values of a matrix is subtly different from other recodes in that the output values are related to the input values in a simple, numerically meaningful way. In fact, a reversal is just a linear transformation Y(i,j)=*X(i,j) + Min(X)-Max(X)*. In order to perform more general transformations of the type Y(i,j) = *bX(i,j) + m* where *b* and *m* are constants, it is necessary to use the **Matrix Algebra** package. For example, to multiply all the values in a matrix called **DAVIS** by 7 you would type (once in **Matrix Algebra**):

-->**davis7 = linear(davis 7 0)**

That means, 'multiply each value of the matrix in dataset **DAVIS** by 7 and then add 0, putting the result in a dataset called **DAVIS7**'. To divide all values by 2 and then subtract 1, type:

-->**davis7 = linear(davis 0.5 -1)**

4.7 Symmetrizing

To symmetrize a matrix X is to force X(i,j) to equal X(j,i) for all *i* and *j*. The need to symmetrize data comes up in a number of contexts. One case is where data are collected on what is logically a symmetric relation, such as "has had lunch with", but due to measurement error (e.g., faulty memories), the actual data are not symmetric. For example Oliver recalls discussing selling arms to Iran with Ronald, but Ronald doesn't recall that. Another case is where an asymmetric relation such as "lent money to" is measured, but the analysis is to be carried out on the more general relation "is financially involved with". In either case, we must reconcile each X(i,j) and X(j,i).

UCINET provides several options for symmetrizing, including replacing both values with the average of the two, or the smallest or the largest. Among the more unusual choices is the possibility of setting both X(i,j) and X(j,i) equal to 1 if X(i,j) = X(j,i) and 0 otherwise. In effect, this creates a new matrix that indicates whether each dyad is in agreement or not. The average of all values of this matrix would then give the proportion of consensual dyads.

4.8 Geodesic Distances and Reachability

A geodesic distance matrix is a matrix of geodesic distances among pairs of nodes. The geodesic distance between two nodes is the number of links in the shortest path between them. On the theory that influence or communication of any kind of one node on another declines with the distance between them, the geodesic distance matrix can be used as an index of influence or cohesion.

To compute a geodesic distance matrix, run the **Network>Cohesion>Geodesics** procedure. In UCINET 6.0, you will find that the notion of geodesic distances has been generalized to handle valued graphs of various types. For example, your data might record the strength of relationship among actors. The strength of a path from node *a* to *b* to *c* to *d* is defined as the strength of the weakest link between them. The geodesic path from any node to any other is defined as the strongest path between them. The geodesic distance between two nodes is defined as the number of links in the strongest path from one to the other. Another kind of valued data consists of costs or distances associated with a pair of nodes. In this case, the geodesic path is defined as the cheapest or shortest path between points, computed by summing the values of each link in the path.

If you run **Distance** you will also find that it gives you options for transforming the distances into proximities or "nearnesses". This is useful when the geodesic distance matrix is being used as a proxy for, say, cohesion or

influence, where larger numbers should mean more cohesion or influence. The simplest option just subtracts the distances from *n* (the number of nodes), much as the **Reverse** procedure would do. Another option takes the reciprocal of distance (1/d(i,j)) and multiplies by maximum possible distance (n-1) so as to rescale the numbers to vary from 0 to 1. An option of theoretical interest (translation: based on questionable assumptions) takes a constant to the power of the distance between two nodes. Thus, if the constant is 0.5, a pair of nodes that is directly connected is scored 0.5, a pair that is two links apart is scored 0.25, a pair three links apart is scored 0.125, and so on. Another option of theoretical interest is one proposed by Burt (1976) and implemented in STRUCTURE. For a given *i* and *j*, it yields a value that is one minus the proportion of actors *i* can reach in the same number of steps *i* can reach *j*. Thus, if *i* and *j* are relatively distant, then the number of actors *i* can reach at that length or less relative to the total number of actors *i* can reach at any length will be high and so the assigned value will be low.

Closely related to distance is the notion of reach. In the case of an adjacency matrix X, this concept usually refers to a transitive closure of the data, such that Y(i,j) = 1 if there exists a path of any kind from *i* to *j*. In UCINET 6.0, however, we have extended the notion to several kinds of valued matrices as well. For example, for a matrix recording strengths, capacities or frequencies of relationships, the reach of *i* to *j* is the strength of the strongest path between them, as defined above. If the strength values are dichotomous 1s and 0s, the strength of the strongest path between any two points is always 1 unless there is no path at all, which yields the familiar notion of reachability. For cost data, reachability is defined as the cost of the cheapest path. For probability data, reachability is defined as the probability of the most probable path between two points. Thus, the reachability matrix can also be used as a measure of cohesion or expected influence.

Other indices of influence or cohesion that have been suggested in the literature are maximum flows, these are computed by procedures listed under **Network>Cohesion>Maximum Flow**.

4.9 Aggregation

A different class of transformations involve aggregating graphs or matrices in various ways. A typical case is combining several relations measured on the same actors. For example, a study might record half a dozen different kinds of ties among members of a group. A rough measure of influence, mutual dependence, or strength of connection between two actors might be the total number of ties (0 to 6) that bind them. Some people call this "multiplexity", but in UCINET we reserve that term for a slightly more sophisticated concept, namely the combination of ties that bind two actors.

There are two ways to add across all relations within a dataset. The first way is to use **data>CSS.** The **CSS** procedure has a number of ways of aggregating. For our purpose here, slice is the right choice. The second way to aggregate within a dataset is through **Matrix>Algebra**. For example, to aggregate across both relations in a dataset called **PADGETT**, type:

-->**agpadge = total(padgett rows cols)**

This command says 'sum the values of dataset **PADGETT**, breaking the results out by rows and columns, and saving the outcome as dataset **AGPADGE**'. To understand how this works, think of **PADGETT** as a 3-dimensional table (like a contingency table) with rows, columns and levels. We want to collapse the table in such a way that the result has only rows and columns. If we had wanted to collapse across columns as well, we could have typed:

-->**rowmarg = total(padgett rows)**

Similarly, to sum all values in the entire dataset, we would use the **total** command without arguments:

-->**bign = total(padgett)**

The **Transform>Multiplex** procedure can be used to reduce a series of matrices in a dataset into a single categorically-valued matrix which assigns a unique value to each unique combination of values across relations. For example, suppose a dataset contained these two matrices:

```
0 1 0 1
0 0 1 1
1 0 0 0
1 1 0 0

0 1 1 0
1 0 1 1
1 0 0 1
0 1 0 0
```

Members of this network exhibit a total of four distinct combinations of relationships (listed in the order they are encountered): not tied on both relations, tied on both relations, not tied on the first but tied on second, and tied on the first but not on the second. A multiplex matrix of these data looks like this:

```
0 1 2 3
2 0 1 1
1 0 0 2
3 1 0 0
```

The single multiplex matrix encodes all the information in both data matrices. A multiplex matrix (or any valued matrix) can be disaggregated by **Transform>Multigraph** procedure. This routine creates a new matrix for each unique value in the input matrix. For example, the multiplex matrix given above would be disaggregated as follows:

```
0 1 0 0
0 0 1 1
1 0 0 0
0 1 0 0

0 0 1 0
1 0 0 0
0 0 0 1
0 0 0 0

0 0 0 1
0 0 0 0
0 0 0 0
1 0 0 0
```

The first matrix indicates which nodes are tied at the "1" level. The second matrix indicates which nodes are tied at the "2" level, and so on. Optionally, the "0" level can also be treated as a valid connection. Also, the output matrices can be made to be cumulative, so that the second matrix would indicate which nodes were tied at level 2 or below.

Another kind of aggregation is the collapsing of nodes into classes or clusters. This is frequently encountered in the context of blockmodeling where the aggregation results in a density matrix. The main output of blockmodeling is a

partition of actors into classes that are homogeneous with respect to some property. It is then desirable to produce a density matrix which gives the average value of ties between actors of one block with actors of another block. For example, if we partition the two matrices above so that the first two actors fall into one class and the second two actors fall in another, we would get the following two density tables:

```
.25 .75
.75  0

.50 .75
.50 .25
```

To obtain a density table, run **Transform>Block**. This procedure takes as input a data matrix and a partition vector. A partition vector indicates the class of each actor in the network. For example, if a set of 10 actors has been partitioned into two classes, the partition vector might consist of this:
{1,2,1,2,2,1,1,2,1,1}. Thus, there is a number for each actor, and each number refers to a class. The partition vector can be typed in from the keyboard or extracted from a previously created dataset (such as produced by CONCOR or CLUSTER).

A more flexible program for collapsing nodes is **Transform>Collapse**. With this routine, you can sum values, average them, or take the minimum or maximum. However, the input is not a partition vector, but rather lists of nodes which are to be collapsed. For example, the partition vector given in the previous paragraph would be represented as follows:

NODES 1 3 6 7 9 10
NODES 2 4 5 8

4.10 Normalizing and Standardizing

Another class of transformations is normalization. This refers to changing the scale of numbers in the data matrix, either for the matrix as a whole, or for each row (or column) separately. For example, one might wish each row of a data matrix to have the same mean and standard deviation. This need arises when each row of the matrix is collected from a different respondent, who may be answering using a personal scale of measurement that is very different from other respondents. For example, if respondents are asked to assess the physical distance from their homes to the homes of all other actors in the network, it may be that some respondents answer in feet, others in yards, and others in meters. In such cases, it is necessary to reduce each row to a common denominator in order to make the data comparable.

The **Transform>Normalize** procedure can normalize with respect to means, marginals, standard deviations, means and standard deviations together, euclidean norms, and maximums. Each type of normalization can be performed on the matrix as a whole, on each row separately, on each column separately, on each row and each column, and on the matrix as a whole. For example, you can make each row have the same mean (0.00) and standard deviation (1.00). Alternatively, you can normalize each column. The **Normalize** procedure even allows you to normalize both rows and columns simultaneously. Still another possibility is to normalize the entire matrix as a whole so that the mean of all numbers is zero, but the mean of any particular row or column might not be.

Normalizing operations can also be done in less automated a fashion using the **Matrix Algebra** package. For example, suppose you have a 30-by-30 matrix in a dataset called **trade** which you would like to have normalized so that columns add to 100. You can do this with the following sequence of commands:

-->**COLSUM = TOTAL(TRADE COLUMNS)**

-->**NTRADE = DIV(TRADE FILL(COLSUM 30 30))**

The first command creates a row vector (actually a 1-by-30 matrix) containing column sums. The next command divides each entry of the **trade** dataset by its column sum. However, since the **DIVIDE** command requires matrices to be the same size, the **FILL** command is invoked first to duplicate the row vector 30 times to create a 30-by-30 matrix.

4.11 Mode Changes

One class of transformations that is quite different from the others discussed in this chapter is the derivation of 1-mode data from 2-mode data. A 1-mode matrix is a matrix in which both rows and columns refer to the exact same objects (typically actors). A 2-mode matrix is one in which rows index one type of object (such as persons) and the columns index another type of object (such as corporate boards). In network analysis, 2-mode data are frequently encountered in the form of affiliation data: information is collected on a set of groups or events that a set of actors is involved with. The data matrix typically has actors for rows and groups or events for columns. A given data value X(i,j) = 1 if actor *i* is associated with group/event *j*, and X(i,j) = 0 otherwise. From this matrix, we typically compute an actor-by-actor matrix by counting the number of groups/events that each pair of actors has in common. The result is a "co-occurrence" or "co-membership" matrix that is then analyzed as ordinary (valued) network data. Alternatively, we can compute an event-by-event matrix that counts the number of actors attending both events for every pair of events, yielding a measure of the overlap in attendance between the events.

The simplest way to generate a 1-mode matrix from a binary 2-mode matrix is to run the **Data>Affiliations** procedure. This procedure takes a 2-mode matrix as input and counts the number of events (or actors) that each pair of actors (or events) are both positively involved in (both have 1s with). It is useful to realize that this process of counting co-occurrence is the same as measuring the similarity of actors (or events) across events (or actors) using a crude, unnormalized similarity measure. Other similarity measures may be equally appropriate for a given analysis, particularly for valued data. To choose from a wider variety of similarity measures, run the **Tools>Similarities** procedure. For example, an interesting choice for binary data would be the exact matches measure, which would count the number of times that, say, two actors attend or do not attend the same events. Thus, not attending the same events is taken into account in evaluating the relationship between the actors. For valued data, the correlation measure might make a good choice. For a dissimilarity measure, such as euclidean distance, run the **Tools>Dissimilarities** program.

A special case of this kind of mode change is the derivation of a 1-mode matrix from a single vector. For example, a network study might collect basic information on the actors, such as age, sex, and income. The question then becomes, How do these attributes relate to the pattern of relationships among the actors?. One way to answer the question is to convert the attribute data into "networks" and use the **QAP** procedure to correlate them. For example, consider the sex attribute, which we will assume is a vector S coded 1 for males and 2 for females. We can create an actor-by-actor matrix which records for each pair of actors whether they are the same sex or not. That is, we assign X(i,j) = 1 if actors i and j are of the same sex and X(i,j) = 0 otherwise. We can then use **QAP** to correlate "is the same sex as" with "is a close friend of". For a metric variable like income, rather than ask whether actors with the same income tend to be friends, we might prefer to ask whether the degree of difference between incomes predicts friendship, in which case we would set X(i,j) equal to the absolute value of the difference between the incomes of actors *i* and *j*.

The simplest way to generate "network" data from individual attribute data is to run **Data>Attribute**. This procedure takes a single attribute as input and generates a square matrix using your choice of three measures (equality, difference, and absolute difference). From the discussion above, it should be clear that the process of converting vectors to matrices or attributes to networks is again precisely the same as computing the similarities

between rows of a 2-mode matrix. The 2-mode matrix in question is the attribute vector, which is simply an n-by-1 matrix in which the rows are actors and the column is a property or attribute.

The reverse process --- generating a 2-mode matrix from a 1-mode matrix --- can also be of interest in a network context. For example, the **Transform>Incidence** procedure converts a node-by-node adjacency matrix into a 2-mode node-by-line matrix in which X(i,j) = 1 if node *i* is incident upon (is touched by) link *j*. This matrix can then be used as an indicator matrix for further analyses.

~ 5 ~
Where is it now?

In this section we provide a list of UCINET IV routines and state what they are called and where they are located in UCINET 6.0. They are ordered as in UCINET IV. If a routine is no longer available we state that it has been deleted. Some routines have been deleted since they have a function that is repeated, or the function has changed under the new operating system in these cases we have named the routine which can be used to perform the same function preceded by the word 'use'.

DOS	
CD	Use File>Change Default Folder
MD	Use File>Create New Folder
DIR	Deleted
COPY	Deleted
PRINT	Deleted
PRINTER	Deleted
EDIT	File>Create/Edit Ascii File
BROWSE	Deleted
SPOOLER	Deleted
SHELL	Deleted
EXIT	File>Exit
DATASETS	
LIST	Deleted
COPY	Deleted
DELETE	File>Delete Ucinet File
RENAME	File>Rename Ucinet File
DESCRIBE	Data>Describe
DISPLAY	Data>Display
SPREADSHEET	Data>Edit
MERGE	Data>Join
TRANSPOSE	Data>Transpose
RESHAPE	Data>Reshape
SORT	Data>Sort
PERMUTE	Data>Permute
EXTRACT	Data>Extract
IMPORT	
DL	Data>Import/Export>Import>DL
RAW	Data>Import/Export>Import >Raw
KRACKPLOT	Data>Import/Export>Import>Krackplot
UCINET3.0	Data>Import/Export>Import>Ucinet 3.0
EXPORT	
DL	Data>Import/Export>Export>DL
RAW	Data>Import/Export>Export> Raw
KRACKPLOT	Data>Import/Export>Export>Krackplot
UCINET3.0	Data>Import/Export>Export> Ucinet 3.0
NETWORKS	

RANDOM	
SOCIOMETRIC	Data>Random>Sociometric
BERNOULLI	Data>Random>Bernoulli
MULTINOMIAL	Data>Random>Multinomial
TRANSFORM	
SORT	Use Data>Sort
PERMUTE	Use Data>Permute
INCIDENCE	Transform>Incidence
LINEGRAPH	Transform>Linegraph
MULTIGRAPH	Transform>Multigraph
SEMIGROUP	Transform>Semigroup
MULTIPLEX	Transform>Multiplex
POOLED	Data>CSS
BLOCK IMAGE	Transform>Block
AFFILIATIONS	Data>Affiliations
ATTRIBUTE	Data>Attribute
CONNECTIONS	
DISTANCE	Network>Cohesion>Distance
REACH	Network>Cohesion>Reachability
FLOW	Network>Cohesion>Maximum Flow
VOLUME	
GEODESICS	Network>Cohesion>Geodesics
ALL PATHS	Deleted
WALKS	Deleted
INFLUENCE	Network>Centrality>Katz (or Hubbell)
DENSITY	Network>Properties>Density
TRANSITIVITY	Network>Properties>Transitivity
CENTRALITY	
DEGREE	Network>Centrality>Degree
CLOSENESS	Network>Centrality>Closeness
BETWEENNESS	Network>Centrality>Betweenness
EIGENVECTOR	Network>Centrality>Eigenvector
FLOW BETWEENNESS	Network>Centrality>Flow Betweenness
INFORMATION	Network>Centrality>Information
BONACICH POWER	Network>Centrality>Power
SUBGROUPS	
COMPONENTS	Network>Regions>Components
K-CORES	Network>Regions>K-Core
CLIQUES	
CLIQUE	Network>Subgroups>Cliques
N-CLIQUE	Network>Subgroups>N-Cliques
N-CLAN	Network>Subgroups>N-Clan
K-PLEX	Network>Subgroups>K-Plex
FACTIONS	Network>Subgroups>Factions
CONNECTIVITY SETS	
BLOCKS	Network>Regions>Bi-Copmonent
LAMBDA SETS	Network>Subgroups>Lambda Sets
POSITIONS	
STRUCTURAL EQUIVALENCE	
PROFILE SIMILARITY	Network>Roles&Positions>Structural>Profile
TABU SEARCH	Network>Roles&Positions>Structural>Optimization

CONCOR	Network>Roles&Positions>Structural>CONCOR
CATIJ	Deleted
AUTOMORPHIC	
GEODESIC EQUIVALENCE	Deleted
MAXSIM	Network>Roles&Positions>Exact>MaxSim
TABU SEARCH	Network>Roles&Positions>Exact>Optimization
ALL PERMUTATIONS	Network>Roles&Positions>Automorphic
REGULAR EQUIVALENCE	
CATEGORICAL	Network>Roles&Positions>Maximal Regular>CATREGE
CONTINUOUS	Network>Roles&Positions>Maximal Regular>REGE
TABU SEARCH	Network>Roles&Positions>Maximal Regular>Optimization
INTERNETWORK	
QAP-CORRELATION	Tools>Statistics>Matrix(QAP)>QAP Correlation
QAP-REGRESSION	Tools>Statistics>Matrix(QAP)>QAP Regression
STOCHASTIC	
P1	Tools>Statistics>P1
MATRICES	
RANDOM	Data>Random>Matrix
PLOT	
SCATTERPLOT	Tools>Scatterplot>Draw
CLUSTER DIAGRAM	Deleted
TRANSFORM	
SYMMETRIZE	Transform>Symmetrize
DICHOTOMIZE	Transform>Dichotomize
RECODE	Transform>Recode
REVERSE	Transform>Reverse
NORMALIZE	Transform>Normalize
COLLAPSE	Transform>Collapse
DIAGONAL	Transform>Diagonal
PARTITION TO SETS	Data>Sets
ALGEBRA	Tools>Matrix Algebra
UNIVARIATE	Tools>Statistics>Univariate
MULTIVARIATE	
SCALING	
MDS	
NON-METRIC	Tools>MDS>Non-Metric
METRIC	Tools>MDS>Metric
SVD	Tools>2-Mode Scaling>SVD
CORRESPONDENCE	Tools>2-Mode Scaling>Correspondence
CLUSTERING	
HIERARCHICAL	Tools>Cluster>Hierarchical
TABU SEARCH	Tools>Cluster>Optimization
REGRESSION	Use Tools>Statistics>Vector>Regression
SIMILARITIES	Tools>Similarities
DISSIMILARITIES	Tools>Dissimilarities
OUTPUT	
PRINT	Use Print Buttons on Output Screens
COPY	Use Save Buttons on Output Screens
APPEND	Deleted

REVIEW	Use File>Edit Previous Log File
OPTIONS	
DIRECTORIES	Deleted
PAGE SIZE	Options>Page Size
OUTPUT	Options>Output Log File Append/Overwrite
PRINTER	Deleted