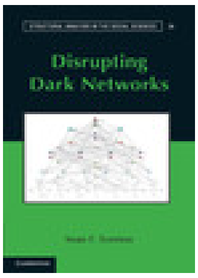


Disrupting Dark Networks

Sean F. Everton

Book DOI: http://dx.doi.org/10.1017/CBO9781139136877

Online ISBN: 9781139136877

Hardback ISBN: 9781107022591

Paperback ISBN: 9781107606685

Chapter

6 - Cohesion and Clustering pp. 170-205

Chapter DOI: http://dx.doi.org/10.1017/CBO9781139136877.010

Cambridge University Press

6

Cohesion and Clustering

6.1 Introduction

A major focus of social network analysis is to identify dense clusters of actors "among whom there are relatively strong, direct, intense, and/or positive ties" (Wasserman and Faust 1994:249). Social network analysts often refer to these clusters of actors as cohesive subgroups and generally assume that "social interaction is the basis for solidarity, shared norms, identity, and collective behavior, so people who interact intensively are likely to consider themselves a social group" (de Nooy, Mrvar, and Batagelj 2005:61). Social network analysts use several approaches for identifying cohesive subgroups. One way is to cluster actors based on attributes (e.g., race, gender), but the more common approach is to focus on the pattern of ties (i.e., relations) among actors. Perhaps not surprisingly, social network analysts have developed a variety of algorithms for identifying subgroups:

> Once analysts began to try to formalize the idea of the clique and to devise mathematical measures of the number and cohesion of cliques, it was . . . recognized that there were a number of different ways of operationalizing the apparently simple idea of the "clique": for example, cliques could be seen as groups of mutually connected individuals or as pockets of high density. Thus, a number of different theoretical models of subgroups emerged, variously described as "cliques," "clusters," "components," "cores," and "circles." Apart from beginning with the letter "c," these concepts have very little in common with one another. (Scott 2000:100)

This chapter explores some (but not all) of the various approaches for using patterns of ties for identifying cohesive subgroups within social

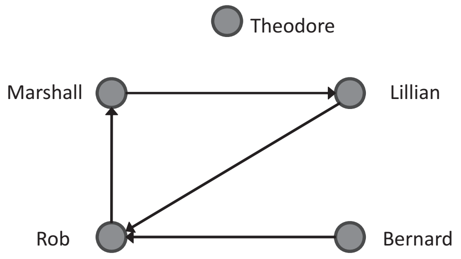


Figure 6.1. Simple Unconnected Directed Network (from De Nooy et al. 2005:66)

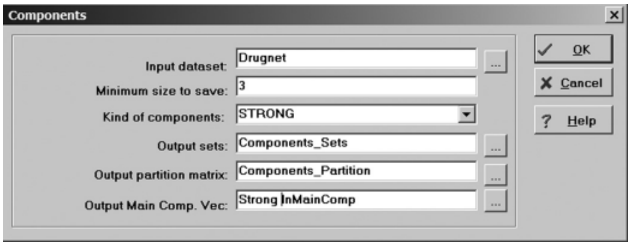
networks.[1] Specifically, it examines components, cores, factions, and Newman groups. It does not consider perhaps the strictest algorithm for identifying subnetworks, namely, "cliques," which are defined as a subset of actors wherein each actor is directly connected to all other actors (in other words, the density of a clique is 100 percent). Cliques are not considered for two reasons: One is because the assumption that each member of a subgroup has a tie to every other member is probably unreasonable. The second is that actors in a network can belong to more than one clique, which makes identifying distinct subgroups often impossible. We briefly mention the algorithm here, though, because as we will see further on, ORA uses a count of the number of cliques to which an actor belongs as a proxy of that actor's centrality.

6.2 Components

Probably the simplest forms of subgroups are components, which are subnetworks in which members are connected (either directly or indirectly) to one another but are not to members of other subnetworks (Hanneman and Riddle 2005). In directed networks, you can identify two types of components: strong and weak. Strong components take into consideration the direction of ties, whereas weak components do not. In a strong component each pair of actors is connected by a (directed) path and no other actor can be added without destroying its connectedness. By contrast, in a weak component each pair of actors is connected by an undirected path (i.e., a semipath) and no other actor can be added without destroying its connectedness.[2] Take, for example, the network in Figure 6.1 (adapted from de Nooy et al. 2005:66). The network contains three

[1] See Hanneman and Riddle (2005, 2011) for an exhaustive review of the various cohesive subgroup algorithms available in UCINET.

[2] Recall that a path is a walk (i.e., a sequence of actors and ties) in which no actor between the first and last actor of the walk occurs more than once (see Chapter 1).


Components

Input dataset: Drugnet
Minimum size to save: 3
Kind of components: STRONG
Output sets: Components_Sets
Output partition matrix: Components_Partition
Output Main Comp. Vec: StrongInMainComp

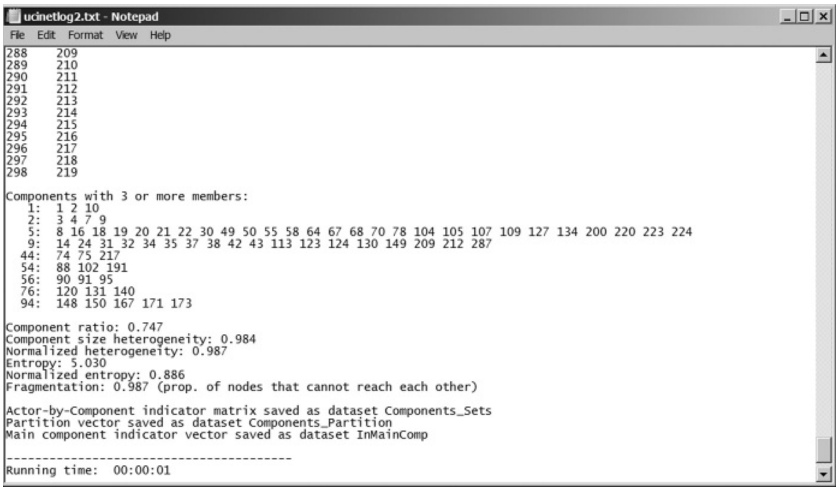
Figure 6.2. UCINET Components Dialog Box

strong components – (1) Bernard; (2) Theodore; and (3) Rob, Lillian, and Marshall – and two weak components – (1) Theodore and (2) Bernard, Rob, Lillian, and Marshall. Bernard, Rob, Lillian, and Marshall do not constitute a strong component because if you follow the directions of the arrows, you cannot "walk" from Rob, Lillian, or Marshall to Bernard. They do, however, constitute a weak component because anyone can walk to anyone else if the direction of the arrows is ignored. (Theodore, of course, cannot reach any of the other actors and is not part of the weak component.) In short, then, strong connectedness is more restrictive than weak connectedness. Thus, although each strongly connected network is also weakly connected, a weakly connected network is not necessarily strongly connected. Figure 6.1 also illustrates that weak components within a given network are (by definition) isolated from one another (unless, of course, there is only one), which suggests that attempting to identify components in a well-connected and undirected network may not prove to be a good use of one's time. Instead, a quick glance at a network map will do the trick. However, as we will see in the next chapter, when faced with a disconnected network, one approach to estimating closeness centrality is to first extract a network's main component (i.e., its largest weak component), calculate closeness centrality for actors included in the main component, and then assign scores of 0.00 to all other actors in the network.

Identifying Components in UCINET

We will begin by examining a directed (i.e., asymmetric), dichotomous network (`Drugnet.##h`) that comes with UCINET and is from a study of the needle-sharing habits of 293 drug users that were recruited through street outreach and a personal drug-user network referral in Hartford, Connecticut (Weeks et al. 2002). To detect components in UCINET we use the *Network>Regions>Components>Simple graphs* command, which brings up a dialog box (Figure 6.2). Because the drug-user network is directed, under the *Kind of components* drop-down menu, we have selected the "Strong" option and changed the name of three of the

[UCINET] Network >Regions >Components >Simple graphs



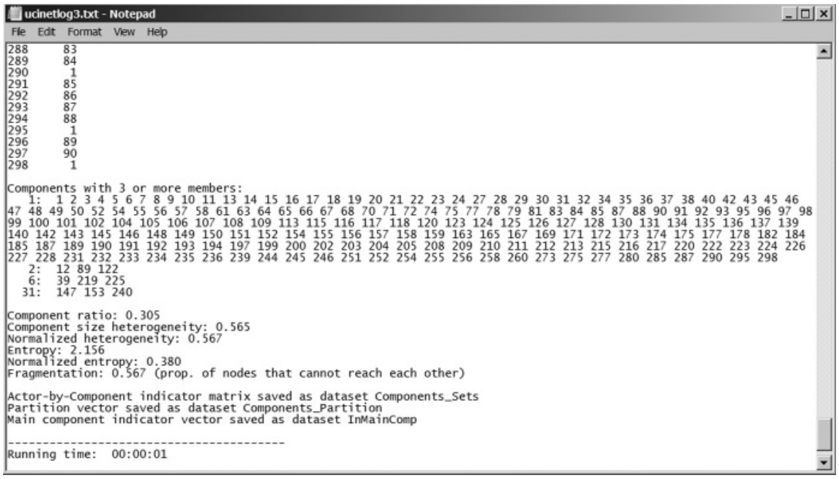
```
288     209
289     210
290     211
291     212
292     213
293     214
294     215
295     216
296     217
297     218
298     219

Components with 3 or more members:
   1:   1 2 10
   2:   3 4 7 9
   5:   8 16 18 19 20 21 22 30 49 50 55 58 64 67 68 70 78 104 105 107 109 127 134 200 220 223 224
   9:   14 24 31 32 34 35 37 38 42 43 113 123 124 130 149 209 212 287
  44:   74 75 217
  54:   88 102 191
  56:   90 91 95
  76:   120 131 140
  94:   148 150 167 171 173

Component ratio: 0.747
Component size heterogeneity: 0.984
Normalized heterogeneity: 0.987
Entropy: 5.030
Normalized entropy: 0.886
Fragmentation: 0.987 (prop. of nodes that cannot reach each other)

Actor-by-Component indicator matrix saved as dataset Components_Sets
Partition vector saved as dataset Components_Partition
Main component indicator vector saved as dataset InMainComp

----------------------------------------
Running time:  00:00:01
```

Figure 6.3. UCINET (Strong) Components Output Log

output files. Otherwise, UCINET's defaults have been accepted, including the option to only detect components of size 3 or greater because components of less than three actors are relatively uninteresting. Note that the dialog box indicates that UCINET produces four different output files,[3] two of which will concern us here: (1) the partition matrix and (2) the main component vector. The "partition matrix" indicates the component of which each actor is a part (if at all – remember we accepted UCINET's default of setting the minimum component size to 3). We will use this file for visualization purposes in NetDraw. The main component vector is another partition that we can use to identify which actors belong to the largest (i.e., main) component, which we can use for visualization purposes as well.

Clicking "OK" yields an output log that first lists the various components and their sizes (including components with sizes less than the minimum size indicated in the dialog box). It then lists the components to which each of the actors (i.e., nodes) belongs (again including components with sizes less than the indicated minimum size). Finally, it lists the components that are the minimum size (in this case three) or larger. This portion of the output log is shown in Figure 6.3. Note that the output indicates that there are nine strong components of size three or greater, with the largest having twenty-seven members (#5) and the second largest having eighteen members (#9). Notice that the output also includes a measure of fragmentation, which, as you will recall, we discussed in Chapter 5.

[3] Interestingly, the output log indicates that the output only includes three files. It does not appear that UCINET currently outputs the fourth file indicated by the dialog box.



ucinetlog3.txt - Notepad

File Edit Format View Help

```
288      83
289      84
290       1
291      85
292      86
293      87
294      88
295       1
296      89
297      90
298       1

Components with 3 or more members:
   1:  1 2 3 4 5 6 7 8 9 10 11 13 14 15 16 17 18 19 20 21 22 23 24 27 28 29 30 31 32 34 35 36 37 38 40 42 43 45 46
47 48 49 50 52 54 55 56 57 58 61 63 64 65 66 67 68 70 71 72 74 75 77 78 79 81 83 84 85 87 88 90 91 92 93 95 96 97 98
99 100 101 102 104 105 106 107 108 109 113 115 116 117 118 120 123 124 125 126 127 128 130 131 134 135 136 137 139
140 142 143 145 146 148 149 150 151 152 154 155 156 157 158 159 163 165 167 169 171 172 173 174 175 177 178 182 184
185 187 189 190 191 192 193 194 197 199 200 202 203 204 205 208 209 210 211 212 213 215 216 217 220 222 223 224 226
227 228 231 232 233 234 235 236 239 244 245 246 251 252 254 255 256 258 260 273 275 277 280 285 287 290 295 298
   2:  12 89 122
   6:  39 219 225
  31:  147 153 240

Component ratio: 0.305
Component size heterogeneity: 0.565
Normalized heterogeneity: 0.567
Entropy: 2.156
Normalized entropy: 0.380
Fragmentation: 0.567 (prop. of nodes that cannot reach each other)

Actor-by-Component indicator matrix saved as dataset Components_Sets
Partition vector saved as dataset Components_Partition
Main component indicator vector saved as dataset InMainComp

----------------------------------------
Running time:  00:00:01
```

Figure 6.4. UCINET (Weak) Components Output Log

What happens if we treat the network as an undirected network by indicating that we want weak components rather than strong? Figure 6.4 displays the answer. When the direction of the network is ignored, UCINET detects only four components of size 3 or more. Note also that the fragmentation score is much lower. That is because the direction of the ties is not taken into account when the score is being calculated, so more pairs of actors can reach one another when they do not have to pay attention to the direction of the arrows.

These two examples help illustrate the fact that the components routine is much better at identifying subgroups when it works with directed rather than undirected data. You can see this for yourself with the Noordin data. Try the *Network>Regions>Components>Simple graphs* command with the `Combined Network (Dichotomized).##h` Noordin data file. You will discover that there is only one component of size 3 or greater, regardless of whether you use the "Strong" or "Weak" option. This illustrates the value of collecting directed data whenever possible (e.g., e-mail and phone communication, the flow of financial and other types of resources).

Network >Regions >Components >Simple graphs

Visualizing Components in NetDraw

Now let us see how NetDraw can help us visualize components. In NetDraw first open the `Drugnet.##h` network file using NetDraw's *File>Open>Ucinet dataset>Network* command. Next, open the `Drugnet Strong Components_Partition.##h`, `Drugnet Strong InMainComp.##h`, `Drugnet Weak Components Partition.`

[NetDraw] File>Open >Ucinet dataset >Network

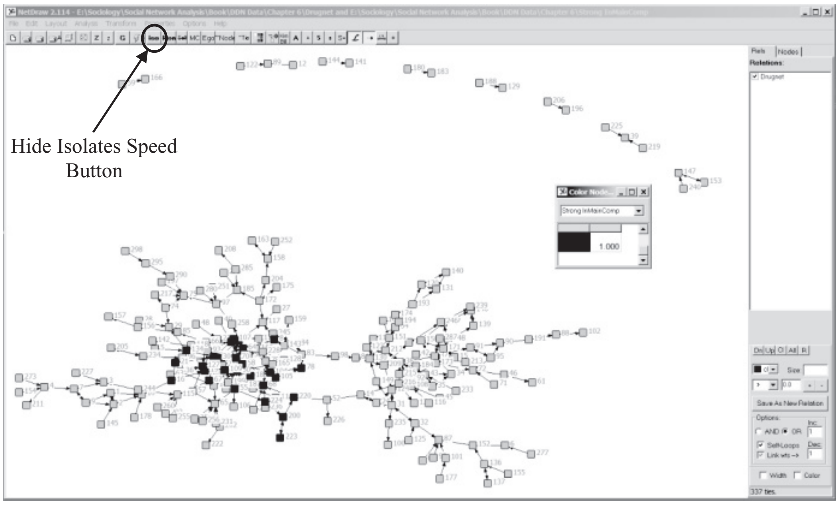


Figure 6.5. NetDraw Visualization of the Drug-User Network's Main Component

`##h`, and `Drugnet Weak InMainComp.##h` files using NetDraw's *File>Open>Ucinet dataset>Attribute data* command. Finally, use the *Properties>Nodes>Symbols>Color>Attribute-based* command so that NetDraw assigns each actor a different color based on the component to which it belongs (i.e., choose the "Strong Components" attribute). This should yield a very colorful picture (not shown) that may not be all that helpful. You can improve the picture somewhat by hiding the isolates by clicking on the "Hide isolates" speed button (see Figure 6.5). Note that the strong components partition generated by UCINET includes more than the nine components of size 3 or greater. It includes all of the components identified by UCINET. Thus, it is hard to visually distinguish the nine largest components in the network. We can, however, identify the largest (i.e., the main) component quite easily by assigning colors to actors using the "Strong InMain Component" attribute (after issuing the *Properties>Nodes>Symbols>Color>Attribute-based* command). This will yield a picture similar to Figure 6.5 where the black-colored actors belong to the main component.

File>Open >Ucinet dataset >Attribute data

Properties>Nodes >Symbols>Color >Attribute-based

Next, assign colors to the nodes using the "Weak Components" attribute. It should yield a network map similar (here, we have converted the colors to gray scale) to Figure 6.6, where the lower portion of the network is the same color because it is a single (weak) component. Every actor can "travel" to every other actor as long as the direction of the ties is ignored. If you were to select the "Weak InMain Component" attribute, you should not be surprised to discover that this is also the main component when tie direction is ignored. Note that the network's

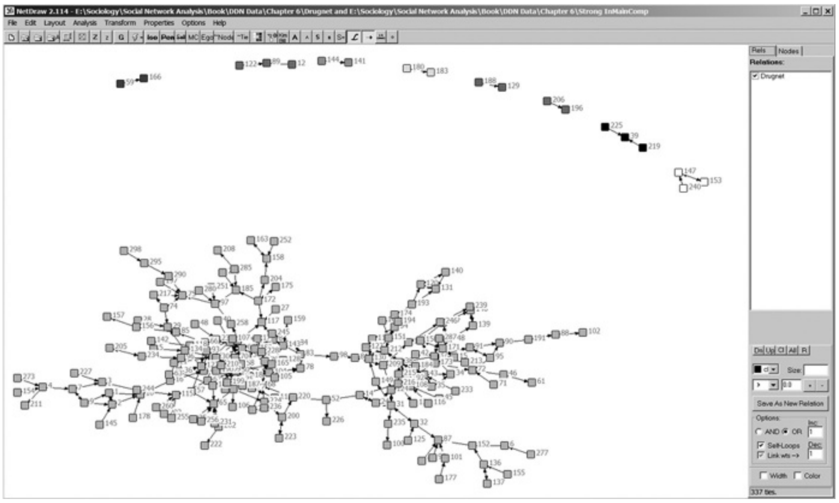


Figure 6.6. NetDraw Visualization of the Drug-User Network's Weak Components

other (and smaller) weak components are assigned their own colors as well.

Analysis >Components

Finally, NetDraw can identify weak components with its *Analysis> Components* command and will produce a network map similar to Figure 6.6. It is helpful to remember, though, that if you are working with directed data and want to identify strong components, then you need to do this in UCINET (or Pajek or ORA) and not NetDraw.

Identifying Components in Pajek

[Pajek's] File >Pajek Project File >Read

Net>Components >Strong, Strong-Periodic, Weak, Bi-Components

Read the Hartford drug network project file (`Drugnet.paj`) into Pajek. Under Pajek's *Net>Components* submenu, you will discover that Pajek includes functions for identifying four types of components: strong, strong-periodic, weak, and bi-components. In this chapter, we will only consider strong and weak components; however, in Chapter 8, in conjunction with examining how to identify brokers and bridges, we will explore bi-components. When you execute the *Strong* or *Weak* command, a dialog box appears that asks for the minimum size of components. As we noted previously, small components are generally uninteresting, so you typically will want to increase the minimum size to three or higher to eliminate isolated actors, which are counted as separate components, and dyads (components of size 2).

The command creates a partition in which each class of actors represents a component (actors that do not belong to a component are assigned to class "0"). You can examine the new partition using the "Edit

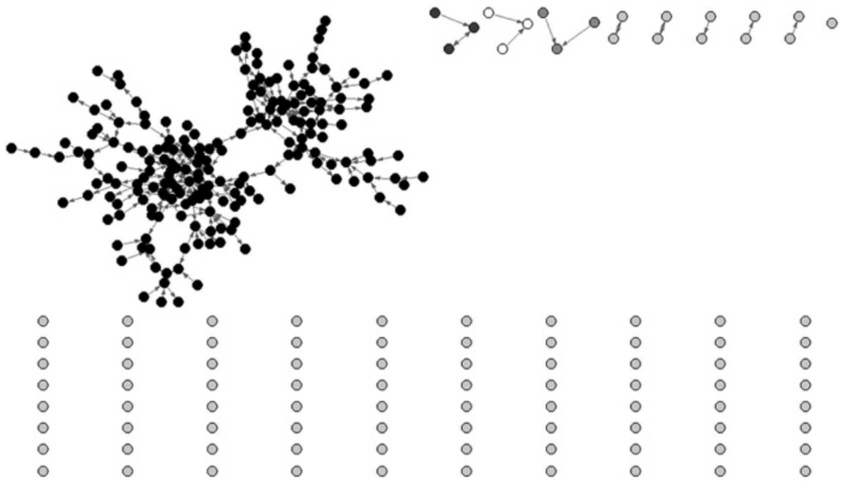


Figure 6.7. Pajek Visualization of the Drug-User Network's Weak Components

Partition" button (located just to the left of the Partition drop-down menu) or the *File>Partition>Edit* command. If you draw the network and partition using the *Draw>Draw-Partition* command, make sure the network and strong components partition are highlighted in the first Network and Partition drop-down menus, respectively. The *Draw>Draw-Partition* command was used to generate Figure 6.7, which is a network map of the drug-user network's weak components. Node color was selected with Pajek's *Options>Colors>Partition Colors>For Vertices* command and then choosing the "Default GreyScale 1" option. The size of the nodes was changed from the default size 4 to 7 using the *Options>Size>of Vertices* command. The components were separated from one another using the *Layout>Energy>Kamada-Kawai>Separate Components* command (you can also use Ctrl-K), which is very useful drawing option in this respect. Finally, the labels were hidden using the *Options>Mark Vertices Using>No Labels* command (you can also use Ctrl-D; Ctrl-L turns the labels back on).

File>Partition >Edit

Draw>Draw-Partition

[Draw Screen] Options>Colors >Partition Colors >For Vertices

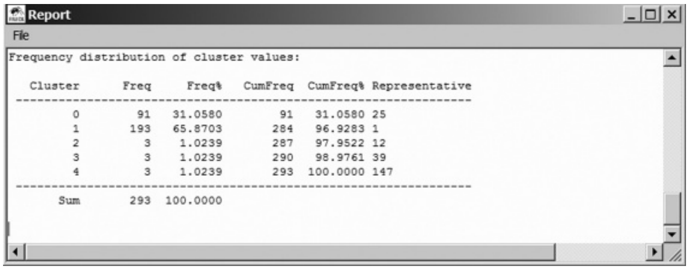
Options>Size >of Vertices

Layout>Energy >Kamada-Kawai>Separate Components

Options >Mark Vertices Using >No Labels

Say you were interested in examining only those users who were a member of the largest weak component. First, with the weak component partition highlighted in the first Partition drop-down menu, use Pajek's *Info>Partition* command (accepting Pajek's defaults), which generates a report similar to the one displayed in Figure 6.8, where you can see that the largest weak component contains 193 actors and has been assigned to class "1" (91 actors have been assigned to class "0," which includes all actors who were members of components of size 2 or smaller). Next, with the drug-user network and weak components partition highlighted

Info>Partition



```
Frequency distribution of cluster values:

   Cluster      Freq      Freq%   CumFreq  CumFreq% Representative
 -----------------------------------------------------------------
         0        91    31.0580        91   31.0580 25
         1       193    65.8703       284   96.9283 1
         2         3     1.0239       287   97.9522 12
         3         3     1.0239       290   98.9761 39
         4         3     1.0239       293  100.0000 147
 -----------------------------------------------------------------
       Sum       293  100.0000
```

Figure 6.8. Pajek Report of Partition Information

Operations >Extract from Network >Partition

Draw>Draw

in their respective drop-down menus, use the *Operations>Extract from Network>Partition* command. Tell Pajek to extract all actors in class one by typing "1" in the resulting dialog box. Click "OK" and Pajek will generate a new network and partition of size 193. You can then use the *Draw>Draw* command to visualize the largest weak component (not shown).

Net >Components >Strong, Weak

Draw>Draw-Partition

Noordin's Network. As we discussed in Chapter 2, one approach to disrupting dark networks involves the use of deception tactics (i.e., PsyOps) that attempt to set network members and/or subgroups against each other. Because trust is such an important resource for dark networks (Carrington 2011; van der Huist 2011), undermining that trust might prove to be an effective strategy. With this in mind, let's examine Noordin's aggregated trust network, which is included in the dichotomized Noordin project file (`Noordin's Network (Dichotomized Edges).paj`). Because it is so well connected, all of the actors, save nine isolates, belong to the same large component. You can see this for yourself by first issuing Pajek's *Net>Components>Strong* or *Weak* command (it does not matter, as the network is undirected), using three as the cutoff size for the smallest component, and then visualizing it with the resulting partition using the *Draw>Draw-Partition* command (Figure 6.9).

Operations >Extract from Network >Partition

This suggests that we may want to drill down and examine a smaller aspect of the network. For example, we might be able to identify meaningful components focusing on Noordin's alive trust network. To do this, we can use the "Current Status (ICG Article)" attribute to extract those network members who were alive (either free or in jail) at the time that the ICG article was published. First, make sure that Noordin's trust network is highlighted in the first or top Network drop-down menu and the current status partition is highlighted in the first or top Partition drop-down menu. Next, extract the alive network using Pajek's *Operations>Extract from Network>Partition* command. Be sure to select clusters (i.e., class) "1–2," because "0" = dead,

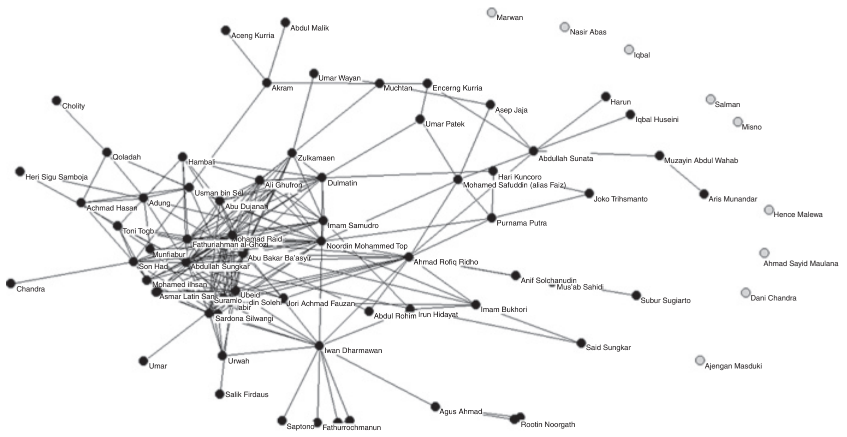


Figure 6.9. Noordin's Combined Network Components (Pajek)

"1" = alive, and "2" = alive but in jail (see Appendix 1). Next, use Pajek's *Net>Components>Strong* or *Weak* command (again using three as the cutoff value) and Pajek will create a new partition. Then visualize the alive network with the new component partition using Pajek's *Draw>Draw-Partition* command. It should look similar to Figure 6.10. Here again, component analysis is not much help. We could continue to drill down to the alive and free trust network, but at that level the network contains very few ties, which makes it difficult to exploit. This is a good example of how social network routines for identifying cohesive subgroups are not always successful, which is why we often have to employ a number of algorithms in our analyses. For now, save your work as a project file, and we will return to it when examining cores.

Net>Components >Weak

Draw>Draw-Partition

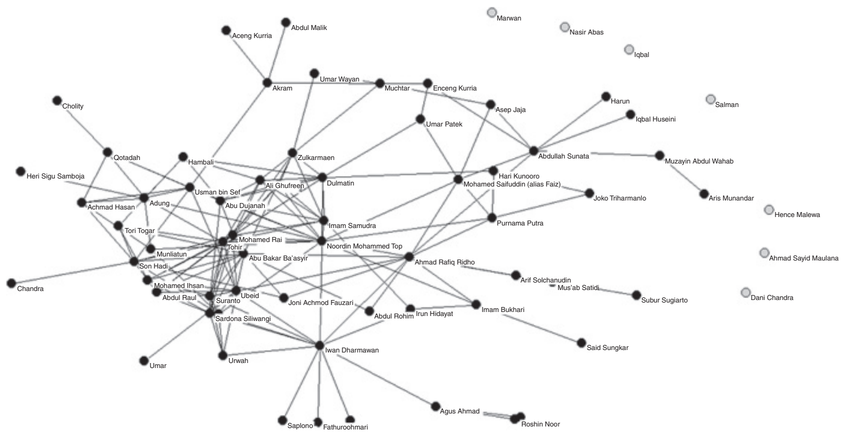


Figure 6.10. Components of Noordin's Alive Trust Network (Pajek)

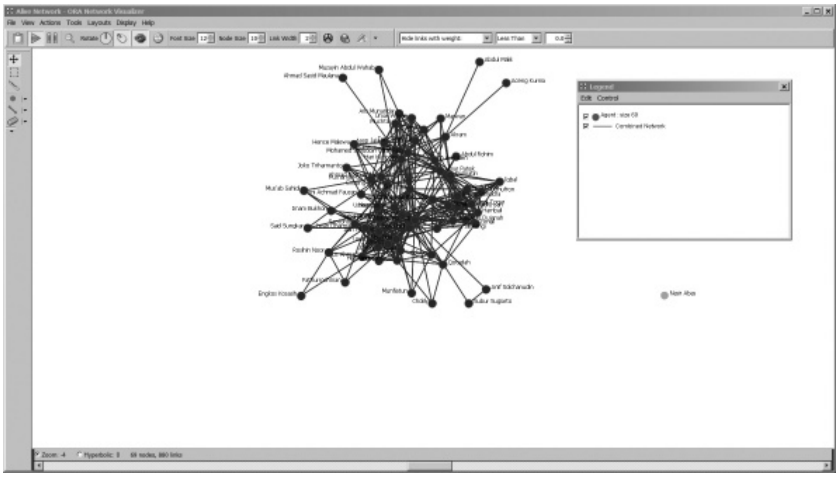


Figure 6.11. Components of Noordin's Alive Combined Network (ORA)

Identifying Components in ORA

There are two ways in ORA to identify components. One is by using ORA's "Local Groups" report that is accessible from ORA's main screen. The other is through the components command found in ORA's visualizer. Because the local groups report not only identifies components but also Newman groups, cliques, and an array of other cluster types, we will postpone our examination of the local groups report until the final section in this chapter. For now, we will examine how to identify components in the ORA visualizer.

[ORA-Main Screen] File>Open Meta-Network

To begin, open the alive Noordin meta-network (`Alive Network.xml`) using ORA's *File>Open Meta-Network* command. Click on the visualize button in the Information/Editor panel. This will open up ORA's visualizer screen. In the Legend box, deselect the combined network ties because these simply repeat the other four types of ties. Next, select the *Display>Node Color>Color Nodes by Component* command, and you should get a network map similar to Figure 6.11 (Note: Grayscale coloring was obtained using the *Display>Grayscale* option). Once again, we see that in this case using the components algorithm is relatively unhelpful in detecting subgroups in the Noordin network.

Display> Node Color>Color Nodes by Component

Display> Grayscale

Table 6.1 summarizes component analysis of each of the four networks plus the combined network in terms of all seventy-nine actors in the dataset, those who are alive (sixty-nine actors), those who are incarcerated (forty-five actors), and those who are alive and free (twenty-four actors). It lists the number of components, the range in size of each component (note that the table includes components of size two), as well as the

Table 6.1. *Components of Noordin's alive network*

Relation	Number of components[4]	Smallest	Largest	Isolates
Trust Network				
Whole (79)	1	70	70	9
Alive (69)	1	62	62	7
Incarcerated (45)	1	37	37	8
Alive and Free (24)	3	2	10	10
Operational Network				
Whole (79)	1	68	68	11
Alive (69)	1	61	61	8
Incarcerated (45)	2	12	29	4
Alive and Free (24)	1	16	16	8
Communications				
Whole (79)	1	74	74	5
Alive (69)	1	63	63	6
Incarcerated (45)	1	33	33	12
Alive and Free (24)	1	14	14	10
Business & Finance				
Whole (79)	4	5	2	64
Alive (69)	4	2	5	57
Incarcerated (45)	3	2	3	36
Alive and Free (24)	0	0	0	24
Combined Network				
Whole (79)	1	78	78	5
Alive (69)	1	68	68	1
Incarcerated (45)	1	44	44	1
Alive and Free (24)	1	19	19	5

number of isolates. In general, component analysis does not do a very good job at identifying distinct subgroups in the various networks. In virtually every case, only one or two components are present, and they are generally visible to the naked eye. That is, they are detectable simply by looking at a network map. The one exception appears to be the business and finance network. This is misleading, however, because the size of each of the components is quite small (2 to 5), which reflects the fact that the network is relatively sparse and disconnected (note how many isolates there are in the network). Thus, in this case, component analysis is relatively unhelpful. Nevertheless, if you were interested in drilling down to (i.e., extracting) a particular subsection of the network (e.g., the largest/main component of the incarcerated portion of the operational network), then component analysis could be a useful first step. Moreover, as we have noted in this chapter and will see in the Chapter 7, extracting the main component is one approach to estimating closeness centrality

[4] The number of components does not include isolates but does include components of size 2 or greater.

when working with a disconnected graph. It is now time to turn our attention to the analysis of cores.

6.3 Cores

As we saw in the previous chapter, degree centrality counts the number of ties of each individual actor and can serve as an alternative measure for the network. We can also use it to identify clusters of actors that are tightly connected through what is known as the *k*-core approach. Formally, a *k*-core is a maximal[5] group of actors, all of whom are connected to some number (*k*) of other group members (de Nooy et al. 2005:70; Hanneman and Riddle 2005, 2011:352; Wasserman and Faust 1994:266–267). In other words, all actors in a 2-core have two or more ties to other actors in the 2-core, all actors in a 3-core have three or more ties to other actors in the 3-core, and so on. This means that higher *k*-cores are nested within lower *k*-cores. Actors in a 4-core are also members of a 3-core, actors in a 3-core are also members of a 2-core, and so on. The reverse is not necessarily true, however. Not all actors in a 2-core are members of a 3-core, and not all members of a 3-core are members of a 4-core. In fact, one way that we use *k*-core analysis to detect cohesive subgroups is by removing the lowest *k*-cores (or extracting the highest *k*-cores) until the network begins to fragment. That said, because higher *k*-cores are nested in lower ones, we generally do not want to use *k*-core analysis to identify actors who are in a position of brokerage between the two groups. Faction analysis (Section 6.3) and Newman's community-detection algorithms (Section 6.4) are better suited for that purpose.

It is important to note that the highest *k*-core in a network does not necessarily correspond to the highest centrality score obtained by an actor in the network. A simple example illustrates this. Take a network in which one actor has a degree centrality of ten but no one else has a degree centrality higher than four. The highest *k*-core will not be a 10-core because every actor in a 10-core would need to be tied to ten other actors, and in this example that is not the case.

Another characteristic about *k*-core analysis is that, although it can identify relatively dense subnetworks, a particular *k*-core is not necessarily a cohesive subgroup in and of itself (de Nooy et al. 2005:71). This is illustrated in Figure 6.12 (created in NetDraw) where the 3-core (black nodes) is composed of two distinct groups, and the 1-core (white nodes) is composed of several different subgroups. Only the 2-core (gray nodes)

[5] The term *maximal* means that no other actor can be added to the cluster without destroying its defining characteristic, which in this case is the minimum number of ties that each actor must have in order to belong to a particular *k*-core.

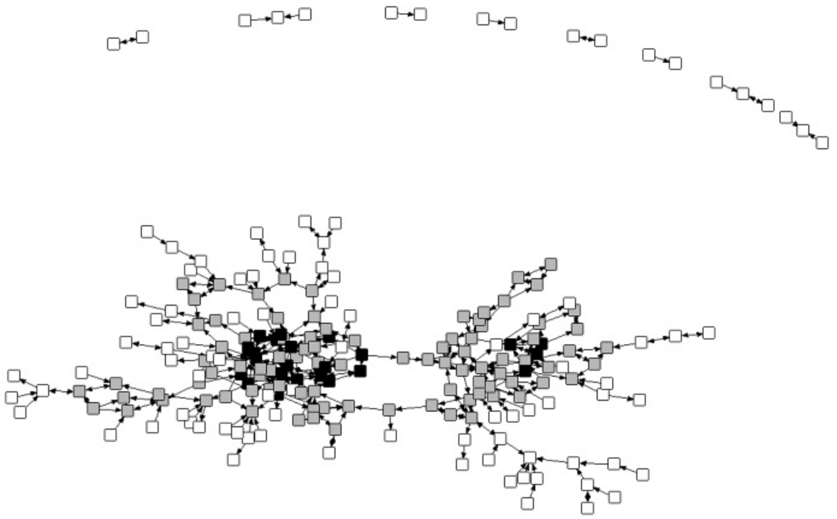


Figure 6.12. *k*-Cores of Drug-User Network (NetDraw)

appears to be connected. This is not necessarily a limitation of *k*-core analysis, however. For example, if we were to extract the 3-core (i.e., remove the 1- and 2-cores), we would be left with two distinct groups, one of size 4 and one of size 19, both of which might lend themselves to further analysis.

Because ORA does not currently include a function for identifying cores, this portion of our analysis is limited to UCINET, NetDraw, and Pajek. Also, to keep things manageable, we will not examine *k*-cores for all of the various networks; instead, we will focus our efforts on the "alive" portions of the trust and operational networks.

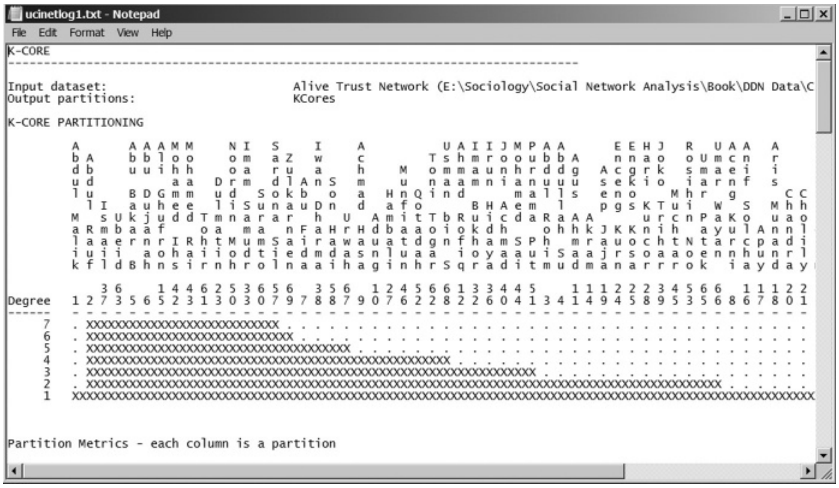
Identifying Cores in UCINET and NetDraw

In UCINET, *k*-cores are detected using the *K-Cores* command found under the *Network>Regions* submenu, which brings up a dialog box (not shown) that asks you to identify the network you wish to analyze. Unfortunately, UCINET only allows us to change the file name of one of the two output files (in this case the default is K-Cores) but not the file we will later import into NetDraw for visualization, which is automatically assigned the name `K-Coreness`. In practice what this means is that we cannot give the file a meaningful name, and (more importantly) it makes it easy to overwrite the file when conducting multiple *k*-core analyses.

[UCINET] Network >Regions >K-Cores

Let's use this command to examine Noordin's alive trust network (`Alive Trust Network.##h`). After selecting OK, UCINET produces an output file similar to the one displayed in Figure 6.13. There are several

Network >Regions >K-Cores



```
K-CORE
--------------------------------------------------------------------------------

Input dataset:                          Alive Trust Network (E:\Sociology\Social Network Analysis\Book\DDN Data\C
Output partitions:                      KCores

K-CORE PARTITIONING

         A       A A A M M     N I   S     I     A           U A I I J M P A A       E E H J   R   U A A   A
         b A     b b l o o     o m   a Z   w     c         T s h m r o o u b b A     n n a o   o U m c n   r
         d b     u u i h h     o a   r u   a     h     M   o m m a u n h r d d g   A c g r k   s m a e i   i
         u d           a a   D r m   d l A n S   m     u   n a a m n i a n u u u   s e k i o   i a r n f   s
         l u     B D G m m   u d   S o k b   o   a   H n Q i n d       m a l l s   e n o     M h r   g       C C
           l I   a u h e e   l i S u n a u D n   d   a f o       B H A e m   l     p g s K T u i   W   S   M h h
         M   s U k j u d d T m n a r a r   h   U   A m i t T b R u i c d a R a A A       u r c n P a K o   u a o
         a R m b a a f     o a   m a   n F a H r H d b a a o i o k d h     o h h k J K K n i h   a y u l A n n l
         l a a e r n r I R h t M u m S a i r a w a u a t d g n f h a m S P h   m r a u o c h t N t a r c p a d i
         i u i i   a o h a i i o d t i e d m d a s n l u a a   i o y a a u i S a a j r s o a a o e n n h u n r l
         k f l d B h n s i r n h r o l n a a i h a g i n h r S q r a d i t m u d m a n a r r r o k   i a y d a y

             3 6     1 4 4 6 2 5 3 6 5 6   3 5 6   1 2 4 5 6 6 1 3 3 4 4 5     1 1 1 2 2 2 3 4 5 6 6   1 1 1 2 2
Degree   1 2 7 3 5 6 5 2 3 1 3 0 3 0 7 9 7 8 8 7 9 0 7 6 2 2 8 2 2 6 0 4 1 3 4 1 4 9 4 5 8 9 5 3 5 6 8 6 7 8 0 1
------   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
     7   . XXXXXXXXXXXXXXXXXXXXXXXXXXX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
     6   . XXXXXXXXXXXXXXXXXXXXXXXXXXXXX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
     5   . XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
     4   . XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX . . . . . . . . . . . . . . . . . . . . . . . . .
     3   . XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX . . . . . . . . . . . . . . . . . . .
     2   . XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX . . . . . .
     1   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


Partition Metrics - each column is a partition
```

Figure 6.13. UCINET *k*-Core Output File (Noordin's Alive Trust Network)

important things to note about the output. First, if the network is asymmetric, UCINET issues a warning that it has symmetrized the network (i.e., it transformed it from a directed to undirected network) and then for each pair of cells assigned cell values based on the maximum value found in the two cells. That is not the case here, but you should be aware of this feature. Next, UCINET's output includes a *k*-core partitioning dendogram that indicates the *k*-core layers to which each actor belongs. Across the top are the names of the actors and down the side is a legend indicating the various *k*-core layers (under the heading, "Degree"). As you can see in Figure 6.13, the largest *k*-core is a 7-core and the smallest is a 1-core. Most actors belong to the 1-core (indicated by the "x" under their name in the bottom row – those that do not belong have a "." under their name), and as the *k*-core level increases, fewer and fewer actors are members. The dendogram also clearly illustrates how members of higher cores are nested in lower cores. Next, if you scroll down the output file, UCINET provides some metrics (not shown) that indicate the proportion of actors per *k*-core; below this you will find a partition that assigns each actor to the highest *k*-core to which it belongs.

[NetDraw] File >Open>Ucinet dataset>Network

If we open the Noordin's alive trust network (`Alive Trust Network. ##h`) and its related *k*-core partition (`K-Coreness. ##h`) – the latter of which is an attribute (not a network) file – in NetDraw using its *File>Open>Ucinet dataset>Network* and *File>Open>Ucinet dataset>Attribute data* commands, respectively, we can then assign node colors based on the *k*-core partition using NetDraw's *Properties>Nodes>Symbols>Colors>Attribute-based* command. This should

File>Open >Ucinet dataset >Attribute data

Properties >Nodes>Symbols >Colors> Attribute-based

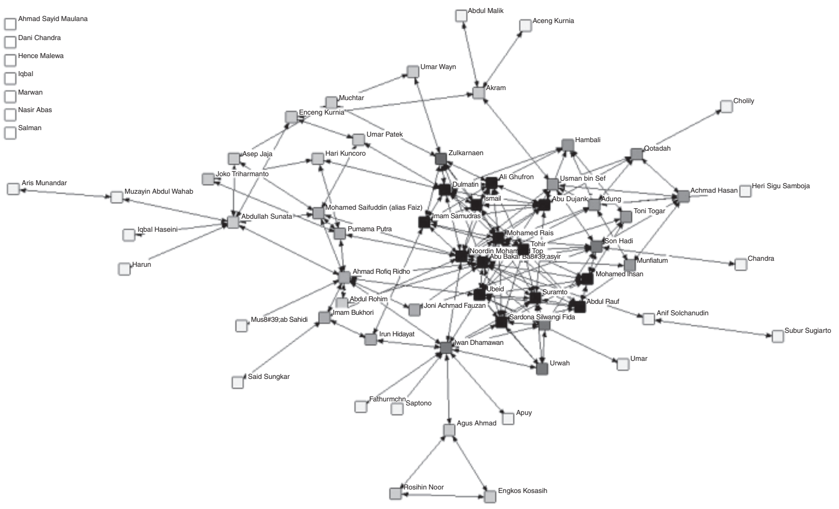
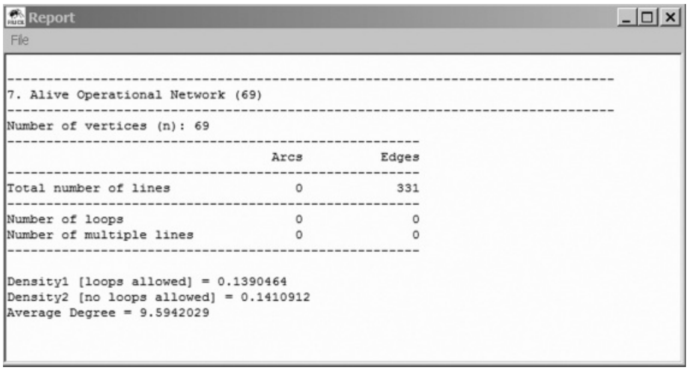


Figure 6.14. *k*-Cores of Noordin's Alive Trust Network (NetDraw)

result in a network map similar to the one shown in Figure 6.14, where the higher the *k*-core layer, the darker the node color. Although with the grayscale coloring it is somewhat hard to distinguish between the cores, what the analysis shows is that there is a well-connected central subgroup (7-core) surrounded by four slightly less connected subgroups (2-core through 5-core – the 6-core consists of one person, who has ties with most of the members of the 7-core).

What might this tell us in terms of strategy? Well, a rehabilitation and reintegration strategy (e.g., a counter-ideology program) would probably not succeed if it targeted 6- and 7-core members because central and well-connected actors are less likely to defect than are peripheral and poorly connected actors (Stark and Bainbridge 1980). Instead, if that is the strategy analysts have chosen to pursue, then they should probably concentrate their efforts on members of the *k*-cores surrounding the 6- and 7-cores. Better yet, because this network includes actors who are both free and in jail, a rehabilitation program that focused on incarcerated actors lying on the network's periphery might be the most cost-effective approach (Mydans 2008). It will be interesting to see if the alive operational network displays similar dynamics. We will examine the *k*-cores of that network using Pajek.[6]

[6] Keep in mind the caveat raised in Chapter 2 concerning rehabilitation programs that seek to reprogram or redirect actors' extremist ideologies: The probability that the reprogramming will "take" in the long run will be increased if the former detainees are prevented from returning to the extremist networks of which they used to be a part, and are instead embedded in networks that support their new ideological orientation (Johnson 2011).


Report
File

```
-----------------------------------------------------------------------------
7. Alive Operational Network (69)
-----------------------------------------------------------------------------
Number of vertices (n): 69
-------------------------------------------------------
                                     Arcs         Edges
-------------------------------------------------------
Total number of lines                   0           331
-------------------------------------------------------
Number of loops                         0             0
Number of multiple lines                0             0
-------------------------------------------------------

Density1 [loops allowed] = 0.1390464
Density2 [no loops allowed] = 0.1410912
Average Degree = 9.5942029
```

Figure 6.15. Pajek Report of General Network Information

Identifying Cores in Pajek

File>Pajak Project File>Read

Noordin's alive operational network is included in the Noordin network project file (`Noordin's Network (Dichotomized Edges).paj`), so the first step is to read this project file into Pajek. After ensuring that the alive operational network is listed in the first Network drop-down menu, we need to check to see whether the network is directed or undirected using the *Info>Network>General* command. This provides a brief output (see Figure 6.15) that indicates the number of vertices (actors), lines, loops, and multiple lines in the network. It also indicates whether the lines are arcs, edges, or a combination of both. As the report indicates, all of the lines in the network consist of edges, which means that it is a symmetric (i.e., undirected) network. If the network included arcs, we would want to convert them to edges using its *Net>Transform>Arc ➔ Edges>All* command because we generally do not take into account tie direction when identifying *k*-cores.

Info>Network >General

Net>Transform >Arc ➔ Edges>All

In Pajek, *k*-cores are detected with the *Core* command found in the *Net>Partitions* submenu. The *Input*, *Output*, and *All* commands distinguish between input cores (based on the number of lines pointing to an actor), output cores (based on the number of lines pointing away from an actor), and cores that ignore the direction of lines. Most of the time we will want to use the *All* command and only to apply it to simple undirected networks. The command creates an "All core partition" that assigns each actor to the highest *k*-core in which it appears. Use the *Info>Partition* command to call up a report on the partition (not shown). The report indicates that largest *k*-core is a 14-core that includes fifteen members (if you think about this, this means that every member of the 14-core is connected to every other member of the core).

Net>Partitions >Cores

Info>Partition

If you visualize the network with the *k*-core partition using the *Draw>Draw-Partition* command, you should get a drawing similar to

Draw>Draw-Partition

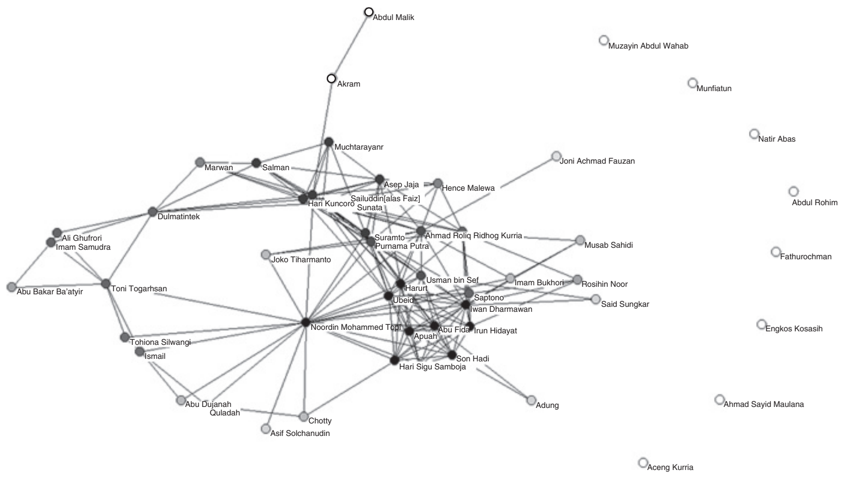


Figure 6.16. Pajek Drawing of Noordin's Alive Operational Network *k*-Cores

the one in Figure 6.16. As the figure illustrates, the alive operational network is structured very differently than the alive trust network. Whereas the *k*-core analysis of the latter found a well-connected core at the center of the network, this analysis detects a series of clusters that appear to center around a single individual: Noordin Mohammed Top. This is perhaps not terribly surprising as it is, after all, Top's network, but it does suggest that the network could be vulnerable to Top's removal or isolation. Of course, the 14-core, of which Noordin is a part, would be unreceptive to most, if not all, rehabilitation and reintegration programs, just like the 7-core of the alive trust network. What analysts could consider is exploring which individuals are on the periphery of both the alive trust and operational networks, and of those, which ones are in jail.

What is clear is that, at least with regards to Noordin's networks, a *k*-core analysis has proved to be more fruitful than the components analysis. That will not always be the case, of course, which again points to the need for analysts to explore networks in a variety of ways in their attempts to get a handle on their underlying structure. Now let us turn to a somewhat different approach to identifying subgroups: faction analysis.

6.4 Factions

Recall that when we looked for components we imagined a "world" where each actor is connected to all other actors in their own subnetwork but have no connections with actors in other subnetworks. What we discovered was that the real-world social networks seldom divide

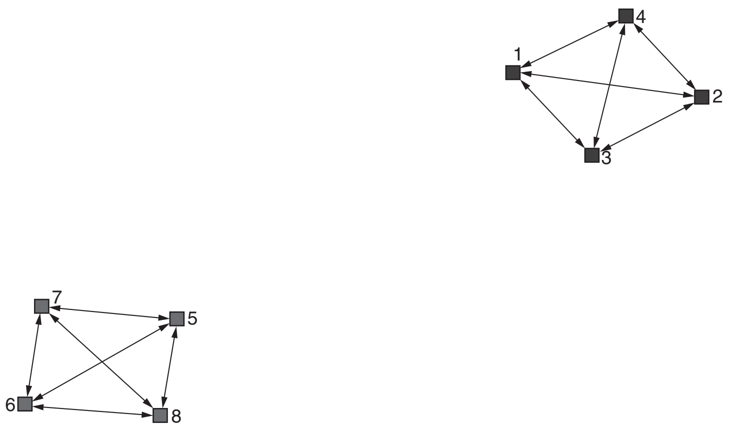
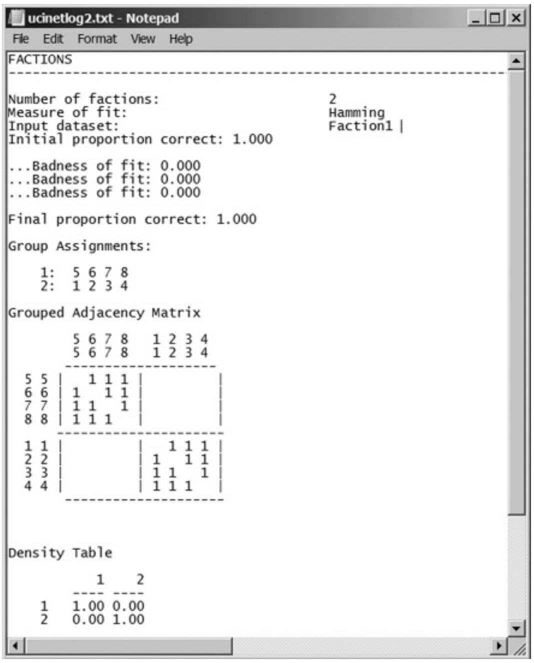


Figure 6.17. A Perfectly Factionalized Network (NetDraw)

themselves as neatly as this. Nevertheless, imagining such a world serves as a useful benchmark for gauging the extent to which a network is "factionalized." That is the approach of faction analysis: It compares an actual network with an idealized one, and then assesses the extent to which the actual network "fits" the idealized one. It is perhaps easiest to illustrate this approach with a couple of simple examples. The above network (Figure 6.17) represents a completely factionalized network. All ties between actors are within their respective subgroups, and there are no ties between actors in different subgroups.

Factional analysis of this network in UCINET yields the output presented in Figure 6.18, of which there are several things worth noting. First, it provides a "badness of fit" score where a lower score is better than a higher score. In this case the score is "0," which means that it is a perfect fit: No ties are absent that should be present, and no ties are present that should be absent. Each time a tie is absent or present when it should not be, it is counted as an error, and the badness-of-fit score is simply a count of such errors. The next measure, "final proportion correct," is a different way of expressing the same thing. Here, the higher the score the better. A grouped adjacency matrix is presented next. Grouped adjacency matrices rearrange the rows and columns so that members of each group are next to one another, and a dashed line indicates the border between the groups. As you can see in this perfectly factionalized network, all possible ties are present within the blocks (i.e., groups) along the diagonal and no ties are present in the off-diagonal blocks. The density table at the bottom of the output is another way of assessing how well the actual network compares to the idealized one. Here, you can see that the density of each block along the diagonal is 100 percent, whereas the density of



```
FACTIONS
----------------------------------------------------------------

Number of factions:                      2
Measure of fit:                          Hamming
Input dataset:                           Faction1 |
Initial proportion correct: 1.000

...Badness of fit: 0.000
...Badness of fit: 0.000
...Badness of fit: 0.000

Final proportion correct: 1.000

Group Assignments:

    1:  5 6 7 8
    2:  1 2 3 4

Grouped Adjacency Matrix

        5 6 7 8   1 2 3 4
        5 6 7 8   1 2 3 4
       -------------------
  5 5 |   1 1 1 |         |
  6 6 | 1   1 1 |         |
  7 7 | 1 1   1 |         |
  8 8 | 1 1 1   |         |
      ---------------------
  1 1 |         |   1 1 1 |
  2 2 |         | 1   1 1 |
  3 3 |         | 1 1   1 |
  4 4 |         | 1 1 1   |
       -------------------


Density Table

           1    2
        ---- ----
    1   1.00 0.00
    2   0.00 1.00
```

Figure 6.18. UCINET Factional Analysis Output

the off-diagonal blocks is 0 percent, which is what we would expect with a perfectly factionalized network.

Now consider the network in Figure 6.19 where two distinct groups clearly exist but one tie is present between the groups and one tie is absent within each of the two groups.

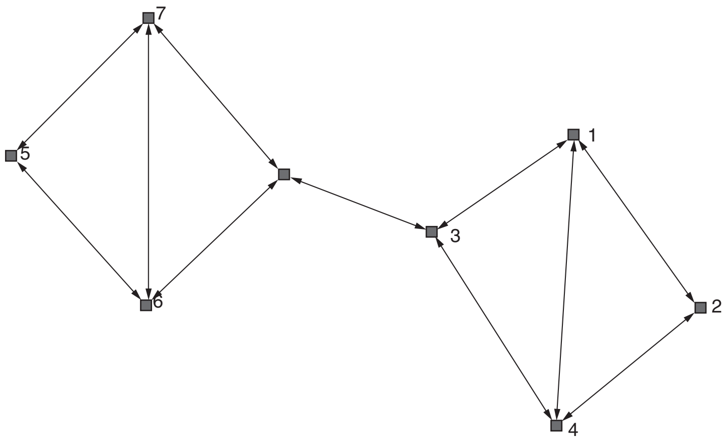
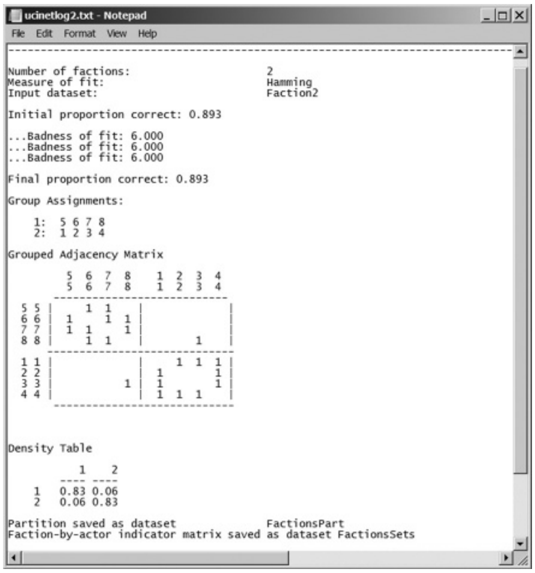


Figure 6.19. An Almost–Perfectly Factionalized Network (NetDraw)

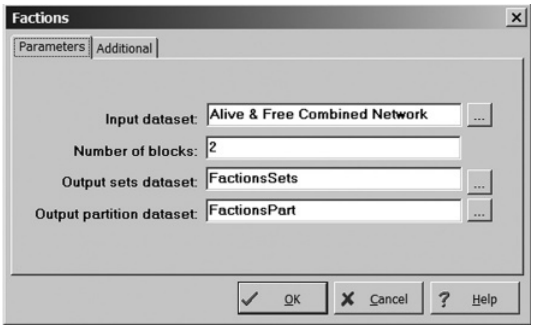


```
--------------------------------------------------------------------------

Number of factions:                             2
Measure of fit:                                 Hamming
Input dataset:                                  Faction2

Initial proportion correct: 0.893

...Badness of fit: 6.000
...Badness of fit: 6.000
...Badness of fit: 6.000

Final proportion correct: 0.893

Group Assignments:

    1:  5 6 7 8
    2:  1 2 3 4

Grouped Adjacency Matrix

         5  6  7  8    1  2  3  4
         5  6  7  8    1  2  3  4
        ---------------------------
  5 5 |     1  1    |             |
  6 6 |  1     1  1 |             |
  7 7 |  1  1     1 |             |
  8 8 |     1  1    |        1    |
       -----------------------------
  1 1 |             |     1  1  1 |
  2 2 |             |  1        1 |
  3 3 |           1 |  1        1 |
  4 4 |             |  1  1  1    |
        -----------------------------


Density Table

          1    2
       ---- ----
    1  0.83 0.06
    2  0.06 0.83

Partition saved as dataset                      FactionsPart
Faction-by-actor indicator matrix saved as dataset FactionsSets
```

Figure 6.20. UCINET Faction Analysis Output

The analysis (Figure 6.20) paints a slightly different picture. The badness-of-fit score is 6 because two ties between actors of different groups are present (i.e., the tie between 3 and 8 is counted twice – see the grouped adjacency matrix in the output) and four ties that should be present within the groups are not (ties between 5 and 8, and 3 and 2 – again counted twice). The presence of these errors lowers the proportion correct to 89.3 percent (50/56) and is illustrated in the grouped adjacency matrix. A quick glance at the density table also tells us that the network is not perfectly factionalized. The density of the blocks along the diagonal is less than 100 percent (.83), whereas the density of the off-diagonal blocks is greater than 0 percent (.06).

How is factional analysis used to detect subgroups? Generally, we use it in an exploratory way, examining the results from several runs where we allow the number of factions to vary in order to see what number yields the best fit (i.e., badness of fit, proportion correct, etc.). Both UCINET and NetDraw implement “faction” algorithms that find the optimal arrangement of actors and measure how well the data actually fit the ideal type. At the time of this writing, ORA does not include one in its toolbox. Pajek does, but we do not cover it here because it is part of its blockmodeling routines, as factions are a special type of block model (Wasserman and Faust 1994:422–423). We will briefly examine Pajek’s approach in Chapter 9, however.



Input dataset: Alive & Free Combined Network
Number of blocks: 2
Output sets dataset: FactionsSets
Output partition dataset: FactionsPart

Figure 6.21. UCINET Faction Dialog Box

Identifying Factions in UCINET and NetDraw

To illustrate factional analysis in UCINET and NetDraw, we will use the combined Noordin alive and free network (`Alive & Free Combined Network.##h`) because faction analysis is easier to illustrate with smaller networks. Also, experience has found that faction analysis appears to work better with smaller networks. We use UCINET's *Network>Subgroups>Factions* command to identify the ideal number of factions in a network. Because we only want UCINET to count the number of errors and not take into account values in each cell of the network matrix, we will want to dichotomize valued networks with the *Transform>Dichotomize* command before proceeding. This is not necessary here because the network has already been dichotomized, but you will want to keep this in mind in the future. If the network contains isolates, which is the case here, we will want to remove them with the *Data>Remove>Remove Isolates* command before running the faction algorithm because UCINET has trouble dealing with isolates. By contrast, NetDraw groups all isolates into a single "faction," and then computes its measures of fit basically ignoring the isolates.

Network >Subgroups >Factions

Transform >Dichotomize

Data>Remove >Remove Isolates

When we issue the *Factions* command in UCINET, it brings up a dialog box similar to the one displayed in Figure 6.21. Note that in addition to asking for the input file, UCINET also asks us to indicate how many "blocks" (i.e., groups) we wish to identify. Here we may be guided by an a priori hunch as to how many subgroups exist (e.g., based on prior analysis), or we may simply engage in an exploratory analysis of the network, varying the number of subgroups and looking for a result that produces the best fit.

Under the Additional tab you will find three "measure-of-fit" options: Hamming, Phi, and Modularity. *Hamming* is the conventional measure of fit and the one illustrated previously in Figures 6.18 and 6.20. *Phi* measures the correlation between the actual data matrix and an idealized one where the density of the diagonal blocks is 100 percent and the density

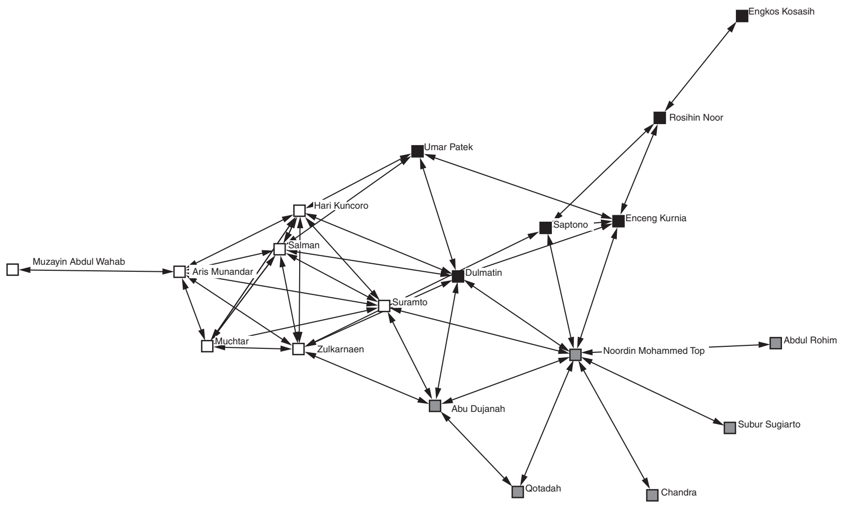
Table 6.2. *Faction analysis of Noordin's alive and free combined network (fit)*

Number of factions	Hamming (proportion correct)	Phi (final correlation)	Modularity (final modularity)
2	0.690	0.411	0.285
3	0.807	0.528	0.319
4	0.842	0.585	0.312
5	0.860	0.602	0.300
6	0.871	0.629	0.284
7	0.877	0.648	0.265
8	0.883	0.669	0.240
9	0.883	0.673	0.219
10	0.883	0.656	0.207
11	0.865	0.620	0.171

of the off-diagonal is 0 percent. *Modularity* is a measure developed by Mark Newman (2006) that compares the ties within and across blocks to what one would expect in a random graph of the same size and having the same number of ties. Formally, it is the fraction of internal ties in each block less the expected fraction, if they were distributed at random but with the same frequency of ties. It is the same measure used to evaluate the fit of Newman groups. All three "measure-of-fit" options produce a similar output, the only difference being the measure of fit that appears at the top of the output.

The results of an analysis of the combined alive network appear in Table 6.2. As you can see, the three measures of fit provide different answers as to what is the optimal number of blocks. According to the conventional Hamming measure, the optimal number of blocks is 8 through 10. Generally, we prefer fewer groups over more, so if we relied solely on this measure, we would probably choose 8 blocks as the optimal number. However, given that the Phi measure identifies 9 blocks as optimal, we may want to consider the latter. When we turn to Newman's modularity score, however, we are presented with a completely different picture. It identifies 3 (possibly 4) blocks as the ideal number.

What should we choose? This is an example of where we should combine metric analysis with visualization. In other words, what number of blocks looks visually correct? While it is impractical to display all ten different options here, Figure 6.22 visually compares what the network looks like broken down into three blocks (i.e., factions, groups) with what it looks like broken down into four. Higher numbers of blocks were also visualized, but the clustering deteriorated as the number of blocks climbed higher than five, suggesting that Newman's modularity score may be the best measure of fit for faction analysis. The difference between the



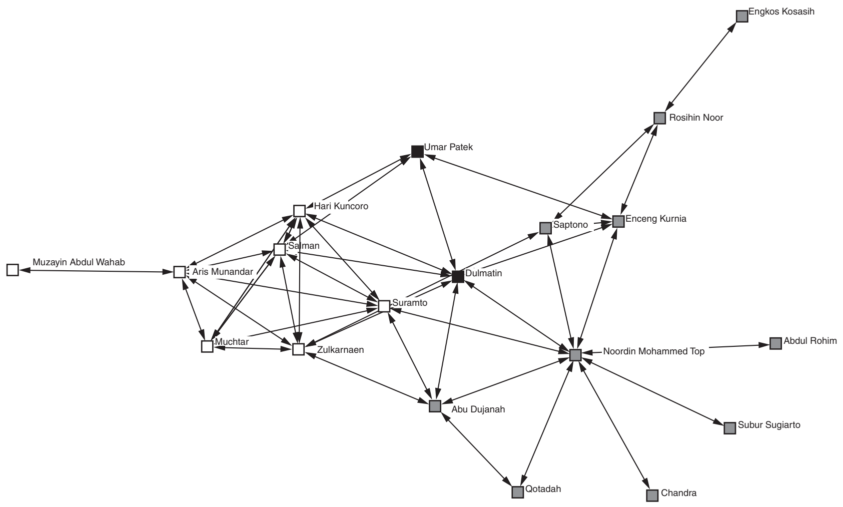
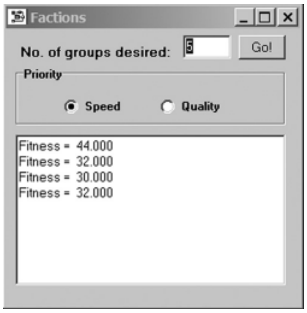


Figure 6.22. Three-Block (top) and Four-Block (bottom) Faction Analyses: Noordin Alive and Free Combined Network (NetDraw)

three-block and four-block network maps is minimal. In the four-block analysis, Dulmatin and Umar Patek are separated from Saptono, Enceng Kurnia, Rosihin Noor, and Engkos Kosasih, whereas in the three-block analysis they are all grouped together. Although one could probably make a good argument for either blocking scheme, at this point the three-block scheme appears to provide a better fit. As always, of course, we would want to check such a conclusion with other forms of intelligence.



[● Speed] ○ Quality

```
Fitness = 44.000
Fitness = 32.000
Fitness = 30.000
Fitness = 32.000
```

Figure 6.23. NetDraw Factions Dialog Box

[NetDraw] File>Open >Ucinet dataset >Network

Analysis >Subgroups >Factions

At this point it is probably best to see whether NetDraw yields similar or completely different results. Open the network (`Alive & Free Combined Network.##h`) with NetDraw's *File>Open>Ucinet dataset>Network* command. To conduct faction analysis in NetDraw, we use its *Analysis>Subgroups>Factions* command, which calls up a dialog box similar to Figure 6.23. Note that you are prompted to indicate the number of groups or blocks you desire to break the network down into, as well as two options for estimating the degree of fit: "Speed" and "Quality." In this case I have chosen the former (although the latter identifies a similar number of ideal blocks) and have estimated measures of fit for a two (44.00), three (32.00), four (30.00), and five (32.00) blocks. Here you can see that NetDraw identifies four factions as the ideal number of groups although the difference between its score and the three and five block scores does not differ substantially. A nice feature of NetDraw is that each time you use it to identify factions it automatically colors the nodes according to their block membership, which facilitates the visual comparison of different blocking schemes. Once again, we are caught between choosing three or four as the ideal number of blocks. The question appears to be whether we should treat Dulmatin and Umar Patek as a separate group or lump them together with Saptono, Enceng Kurnia, Rosihin Noor, and Engkos Kosasih. Let us turn to our next set of clustering algorithms, Newman groups, to see whether it helps us resolve this dilemma.

6.5 Newman Groups

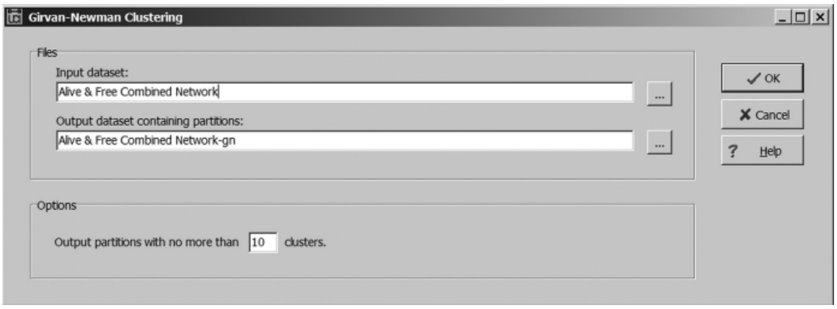
Perhaps the greatest advance in recent years for detecting subgroups are the community-detecting algorithms developed by Mark Newman and his colleagues (Clauset, Newman, and Moore 2004; Girvan and Newman 2002; Newman 2006). Newman's approach is similar to faction analysis in that subgroups are defined as having more ties within and fewer ties between groups than would be expected in a random graph of the same size with the same number of ties. He has developed several algorithms, two of which have found their way into ORA and one (which is one of the two implemented in ORA) that has been implemented in UCINET/NetDraw. Pajek has yet to include any Newman group algorithms, so here we only explore how they are implemented in NetDraw and ORA.

Girvan-Newman (2002) is probably the best-known and most widely implemented Newman group algorithm and is found in UCINET, NetDraw, and ORA. It begins with a connected network and then strategically removes ties, a process that partitions the network into an increasing number of clusters. Each time a new cluster forms (i.e., breaks apart from a larger cluster), the algorithm estimates the network's modularity (discussed previously), and the partition with the highest modularity score is generally considered to represent the optimal partitioning of the network.

Key to the Girvan-Newman algorithm is the notion of edge betweenness. Edge betweenness is similar to (node) betweenness centrality, which we introduced in the first chapter and will take up in more depth in the next. The difference is that edge betweenness estimates the betweenness centrality of edges (i.e., ties) in the network, whereas node betweenness estimates the betweenness centrality of nodes (i.e., actors). Like node betweenness, edge betweenness measures the extent to which each edge in a network lies on the shortest path linking all pairs of actors (i.e., geodesic) in the network. Thus, the more geodesics on which an edge lies, the higher its betweenness score. The fewer geodesics, the lower its betweenness score. In fact, if an edge does not lie on any geodesics, then its betweenness score is zero.

In the Girvan-Newman algorithm, edge betweenness is used to determine the order that edges are removed. "Girvan and Newman reasoned that since there should be relatively few edges linking individuals in different groups, those linking edges should display a high degree of betweenness" (Freeman 2011:32). The algorithm begins by calculating edge betweenness for all ties in the network. It then removes the edge (or edges) with the highest edge betweenness score. After they are removed, the edge betweenness of all the ties is recalculated, and the edge (or edges) with the highest score is removed. This process is repeated until no edges remain. Each time the removal of an edge causes the network to break into separate clusters, the algorithm calculates a modularity score, and, as noted previously, the clustering level that yields the highest score is considered to be the ideal partitioning of the group.

The development of the Girvan-Newman algorithm led to a sudden rush among physicists and computer scientists to develop community-detection algorithms (Freeman 2011:32–33) that were more efficient and quicker. One of the criticisms of the original Girvan-Newman algorithm



Files

Input dataset: Alive & Free Combined Network

Output dataset containing partitions: Alive & Free Combined Network-gn

Options

Output partitions with no more than 10 clusters.

Figure 6.24. UCINET Girvan-Newman Dialog Box

was that it was somewhat slow in detecting communities in networks.[7] Newman (2004) himself agreed, and in response he proposed a "greedy" algorithm that begins by treating each actor as a cluster unto itself, and then "at each stage in the process, the pair of clusters that yields the highest modularity is merged" (Freeman 2011:33). He then teamed with Aaron Clauset and Cristopher Moore to improve its efficiency (Clauset, Newman, and Moore 2004), and it is this algorithm, along with the Girvan-Newman, that is implemented in ORA. Like the initial greedy algorithm, the Clauset, Newman, Moore algorithm begins with each actor being the member of a community of one. It then repeatedly joins together the two communities whose combination produces the largest increase in modularity. It repeats this process until all actors are joined together into a single community. And as with Girvan-Newman, the network partition that yields the highest modularity score is considered to be the optimal partition of the network.

Identifying Newman Groups in UCINET and NetDraw

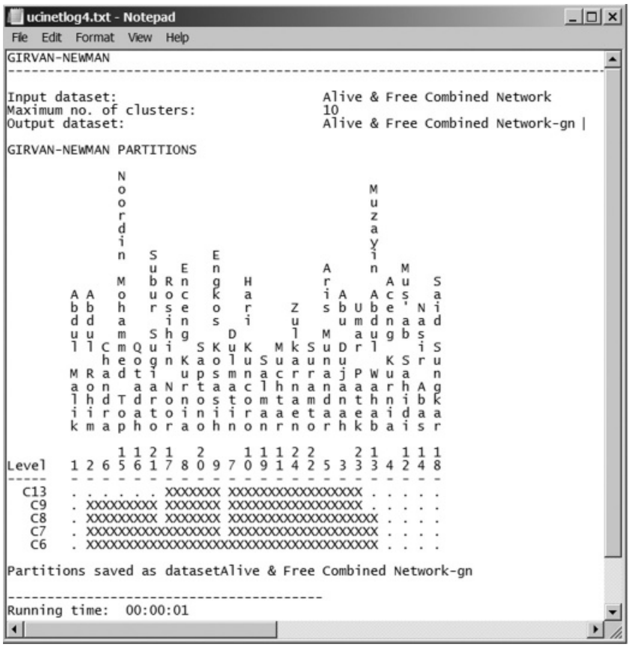
Although both UCINET and NetDraw include the Girvan-Newman algorithm in their toolboxes, NetDraw's implementation is currently superior to UCINET's because it generates a modularity score and UCINET does not.[8] We begin by briefly considering how to use Girvan-Newman in UCINET before moving to NetDraw.

[UCINET] Network >Subgroups >Girvan-Newman

In UCINET the algorithm is accessed with the *Network>Subgroups> Girvan-Newman* command (still using the `Alive & Free Combined Network.##h`), which calls up a dialog box similar to Figure 6.24. Like most UCINET dialog boxes, it asks users to indicate the network that is

[7] *Slow* is a relative term here. When working with networks of the size we examine in this book, the Girvan-Newman algorithm is able to detect the optimum partition within a matter of seconds.

[8] UCINET does, however, include a "Cluster adequacy" function where you can obtain the modularity score (and four other measures of fit) for any network cluster partition.



```
GIRVAN-NEWMAN
------------------------------------------------------------------------

Input dataset:                          Alive & Free Combined Network
Maximum no. of clusters:                10
Output dataset:                         Alive & Free Combined Network-gn

GIRVAN-NEWMAN PARTITIONS

              N
              o                               M
              o                               u
              r                               z
              d                               a
              i                               y
              n   S       E                   i
                  u   E   n             A     n   M
              M   b R n   g   H         r       A u   S
        A A   o   u o c   k   a         i A   A c s   a
        b b   h   r s e   o   r     Z   s b U b e ' N i
        d d   a     i n   s   i     u     u m d n a a d
        u u   m   S h g     D       l   M   a u g b s
        l l C m Q u i   S K u K   M k S u D r l     i S
            h e o g n K a o l u S u a u n u     K S r u
        M R a d t i   u p s m n a c r r a j P W u a   n
        a o n   a a N r t a a c l h n a n a a a r h A g
        l h d T d r o n o s t o m t a m d n t h n i b k
        i i r o a t o i n i i r a a e t a a e a i d a a
        k m a p h o r a o h n o n r n o r h k b a i s r

              1 1 2 1   2     1 1 1 2 2     2 1   1 1 1
Level   1 2 6 5 6 1 7 8 0 9 7 0 9 1 4 2 5 3 3 3 4 2 4 8
-----   - - - - - - - - - - - - - - - - - - - - - - - -
  C13   . . . . . . XXXXXXX XXXXXXXXXXXXXXXXX . . . . .
   C9   . XXXXXXXXX XXXXXXX XXXXXXXXXXXXXXXXX . . . . .
   C8   . XXXXXXXXX XXXXXXX XXXXXXXXXXXXXXXXXXX . . . .
   C7   . XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXX . . . .
   C6   . XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX . . . .

Partitions saved as datasetAlive & Free Combined Network-gn

----------------------------------------
Running time:  00:00:01
```

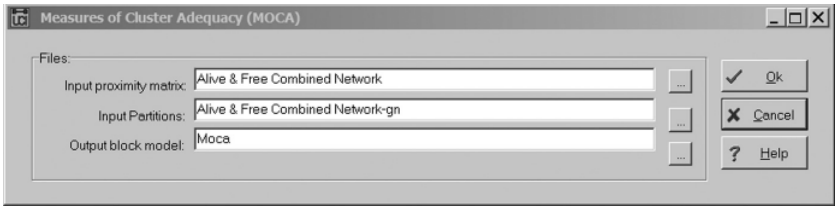
Figure 6.25. UCINET Girvan-Newman Output

to be analyzed, and it "offers" a default name for the output file, which, of course, can be changed. At the bottom of the dialog box, UCINET allows users to choose the maximum number of clusters into which the network will be partitioned.

If we click "OK," UCINET generates an output file (see Figure 6.25) and a partition that we can use to visualize the various clustering levels identified by the algorithm. You can see from the output that UCINET has identified five partitions: one with six clusters (6C), one with seven (7C), one with eight (8C), one with nine (9C), and one with thirteen (13C). But, you might be asking, Didn't we ask UCINET to give us no more than ten clusters? We did, but UCINET treats each isolate as a separate cluster and does not count those clusters as "real" clusters. In other words, the six-cluster partition is really just one cluster (i.e., six clusters less the five isolates), the seven-cluster partition is really two, the eight-cluster, three, and so on.

Although UCINET Girvan-Newman function does not calculate a modularity score for each of the partitions, UCINET's "cluster adequacy" function does. In fact, it provides users with five measures of fit for any combination of network and network partition. This is accessed with the *Tools>Cluster Analysis>Cluster adequacy* command, which calls up a dialog box similar to Figure 6.26.

Tools>Cluster Analysis>Cluster adequacy



Files:
Input proximity matrix: Alive & Free Combined Network
Input Partitions: Alive & Free Combined Network-gn
Output block model: Moca

Figure 6.26. UCINET Cluster Adequacy Function

Here, you see that the network has been loaded in the *Proximity Matrix* drop-down menu, and the related Girvan-Newman partition has been loaded as the clustering partition. When we click OK, UCINET generates five measures of fit (i.e., cluster adequacy). We will not discuss all these here but simply point out that the second line ("Q") contains the modularity score.[9] As you can see, the three-cluster (8C) provides the best fit with the two-cluster (7C) not too far behind. As we will see, these measures agree with those we will obtain using NetDraw (Figure 6.27).

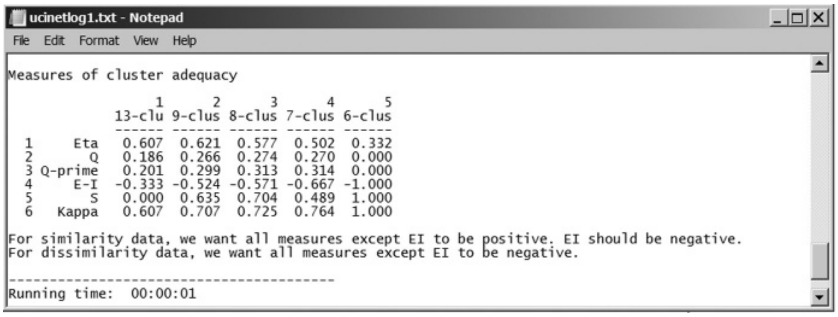
[NetDraw] File >Open>Ucinet dataset >Network

File>Open >Ucinet dataset >Attribute data

Properties >Nodes >Symbols >Color >Attribute-based

To visualize the network and its companion Girvan-Newman partition in NetDraw, we need to read in both Noordin's alive and free combined network (`Alive & Free Combined Network`) into NetDraw and the corresponding partition (`Alive & Free Combined Network-gn`) using the *File>Open>Ucinet dataset>Network* and *File>Open>Ucinet dataset>Attribute data* commands, respectively. Then with NetDraw's *Properties>Nodes>Symbols>Color>Attribute-based* command, we can indicate which partitions we want NetDraw to visualize. Figure 6.28 compares the two-cluster (i.e., C7) and three-cluster (i.e., C8) partitions (the isolates are hidden). The only difference is that in the nine individuals that are treated as a single group in the two-cluster (right-hand side of the graph) are divided into two groups in the three-cluster. Comparing these results with those we obtained using faction analysis (Figure 6.22), we can see that the primary difference is that Dulmatin and Umar Patek are no longer treated as a group unto themselves or grouped with Saptono, Enceng Kurnia, Rosihin Noor, and Engkos Kosasih, but instead are grouped with Hari Kuncoro, Suramto, Salman, Zulkarnaen, Muchtar, Aris Munandar, Muzayin Abdul Wahab, and Abu Dujanah. Abu Dujanah is also placed in a different group with this algorithm than with faction analysis. What are we to make of these conflicting results? One thing they suggest is that Dulmatin and Patek may be their own "group" and that together they act as brokers between the other two groups with which they are occasionally tied. Abu Dujanah may also be in a position of brokerage.

[9] See UCINET's Help function for a description of the other measures of fit.



```
Measures of cluster adequacy

                  1       2       3       4       5
             13-clu 9-clus 8-clus 7-clus 6-clus
             ------ ------ ------ ------ ------
  1     Eta   0.607  0.621  0.577  0.502  0.332
  2       Q   0.186  0.266  0.274  0.270  0.000
  3 Q-prime   0.201  0.299  0.313  0.314  0.000
  4     E-I  -0.333 -0.524 -0.571 -0.667 -1.000
  5       S   0.000  0.635  0.704  0.489  1.000
  6   Kappa   0.607  0.707  0.725  0.764  1.000

For similarity data, we want all measures except EI to be positive. EI should be negative.
For dissimilarity data, we want all measures except EI to be negative.

----------------------------------------
Running time:  00:00:01
```

Figure 6.27. UCINET Cluster Adequacy Output

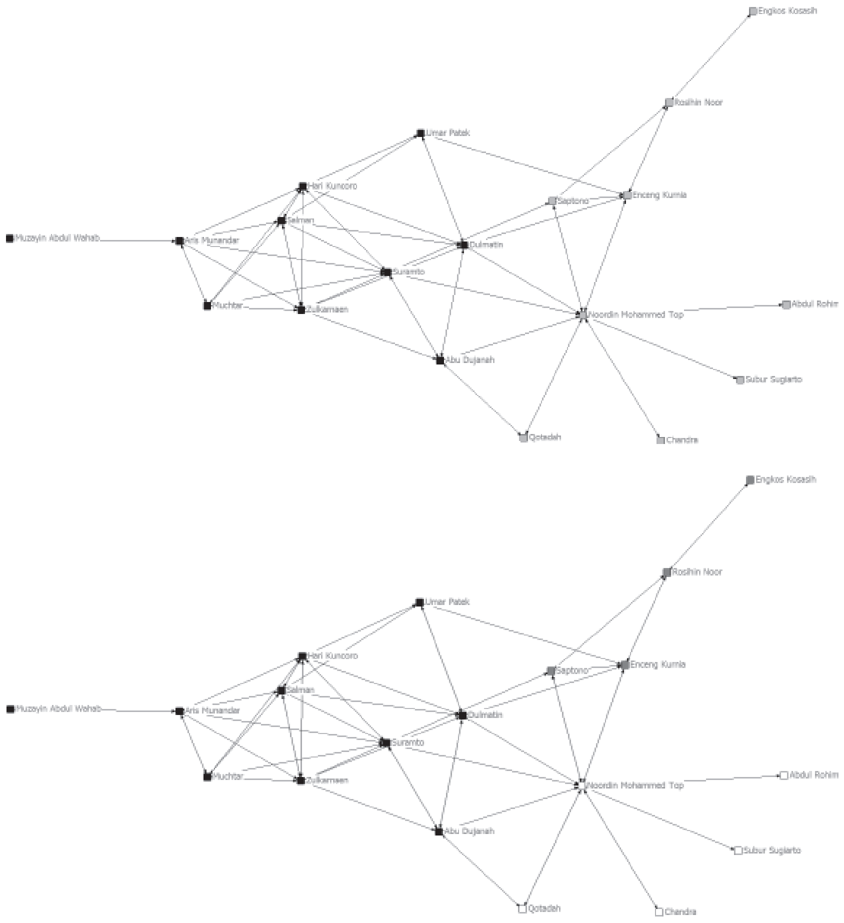
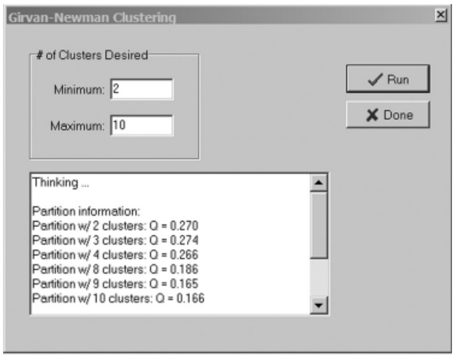


Figure 6.28. Two-Cluster (top) and Three-Cluster (bottom) Girvan-Newman: Alive and Free Combined Network (NetDraw)



```
Thinking ...

Partition information:
Partition w/ 2 clusters: Q = 0.270
Partition w/ 3 clusters: Q = 0.274
Partition w/ 4 clusters: Q = 0.266
Partition w/ 8 clusters: Q = 0.186
Partition w/ 9 clusters: Q = 0.165
Partition w/ 10 clusters: Q = 0.166
```

Figure 6.29. NetDraw Girvan-Newman Dialog Box

Analysis >Subgroups

We can detect Newman groups in NetDraw using its *Girvan-Newman* command, which is located under the *Analysis>Subgroups* menu. It calls up a dialog box similar to Figure 6.29. Notice that NetDraw asks you to indicate the minimum and maximum number of clusters you want to identify. The defaults are 2 and 10 (which are used here), but they are easily changed. Clicking on "Run" generates output in the open box below the cluster option boxes. As you can see, NetDraw identifies the number of clusters (not counting isolates) in each partition along with the associated modularity score ("Q"). It also automatically colors the nodes according to partition with fewest groups, but this can be changed with its *Properties>Nodes>Symbols>Color>Attribute-based* command. NetDraw's analysis agrees with UCINET and indicates that the partition with the highest modularity score is the one with three clusters. Let's see if ORA provides a similar answer.

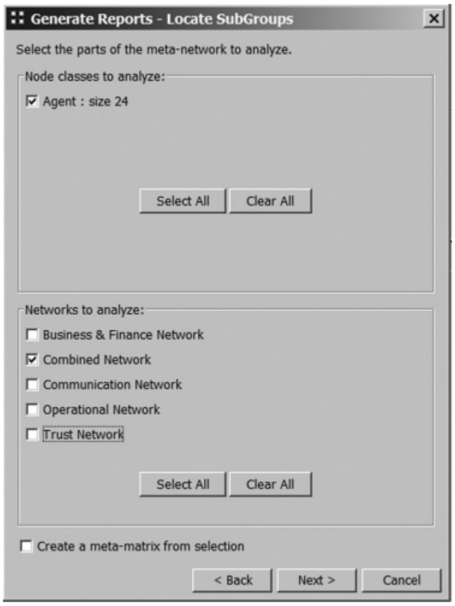
Properties>Nodes >Symbols >Color >Attribute-based

Identifying Newman Groups in ORA

File >Open Meta-Network

Analysis >Generate Reports >Characterize Groups and Networks >Locate SubGroups

In ORA, users can identify Newman groups from either the main screen or the visualizer. First, we need to open the alive and free meta-network (`Alive and Free Network.xml`) using ORA's *File>Open Meta-Network* command. To detect Newman groups from ORA's main screen, we use the "Locate SubGroups" report, which is accessed through the *Analysis>Generate Reports>Characterize Groups and Networks>Locate SubGroups* command. After highlighting the alive and free meta-network in ORA's Meta-Network panel, issue the command and a dialog box (not shown) will appear. After ensuring that the alive and free network box is selected, click "Next," and another dialog box similar to Figure 6.30 should appear.


Generate Reports - Locate SubGroups

Select the parts of the meta-network to analyze.

Node classes to analyze:

☑ Agent : size 24

Networks to analyze:

☐ Business & Finance Network
☑ Combined Network
☐ Communication Network
☐ Operational Network
☐ Trust Network

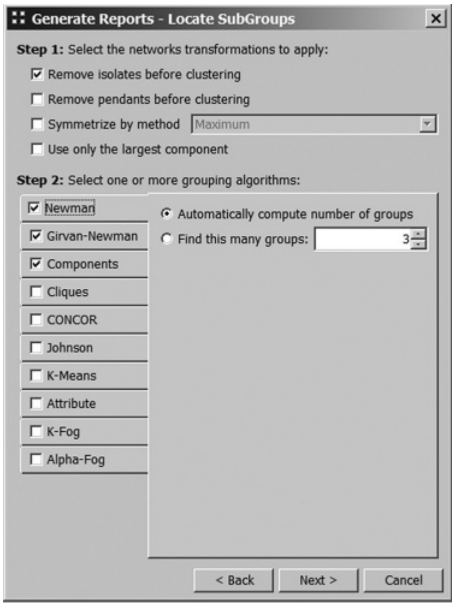
☐ Create a meta-matrix from selection

Figure 6.30. ORA Locate Subgroups Dialog Box

Note that all of the networks in the meta-network are listed. If all of them are selected, however, ORA does not detect clusters for each separate network but instead combines them and identifies clusters based on this combination. Thus, we need to identify the specific network we wish to examine. Because we are focusing on the combined alive and free network, we should deselect all of the boxes except this one (see Figure 6.30). Clicking "Next" again brings up the next dialog box in the series (Figure 6.31), and this illustrates what we noted back in Section 6:1; namely, that with its report-based approach to SNA, ORA allows users to access several different clustering algorithms at the same time and have them appear in a single report.

As you can see, ORA implements two different Newman algorithms (the first one listed is the Clauset, Newman, Moore algorithm), components, cliques (which we discussed briefly at the beginning of the chapter), CONCOR (a type of blockmodeling that we will consider in Chapter 9), and five other clustering techniques that we do not consider in this book.[10] We will typically want to select the "Remove isolates before clustering" option at the top of the dialog box. Also, both of the Newman groups clustering algorithms automatically compute the number of groups (i.e., the ideal partition based on modularity), but if you are working with an a priori theory as to what the number of groups should be or you simply

[10] See ORA's Help function for a description of the other clustering algorithms.



Generate Reports - Locate SubGroups

Step 1: Select the networks transformations to apply:

- [✓] Remove isolates before clustering
- [ ] Remove pendants before clustering
- [ ] Symmetrize by method: Maximum
- [ ] Use only the largest component

Step 2: Select one or more grouping algorithms:

- [✓] Newman
- [✓] Girvan-Newman
- [✓] Components
- [ ] Cliques
- [ ] CONCOR
- [ ] Johnson
- [ ] K-Means
- [ ] Attribute
- [ ] K-Fog
- [ ] Alpha-Fog

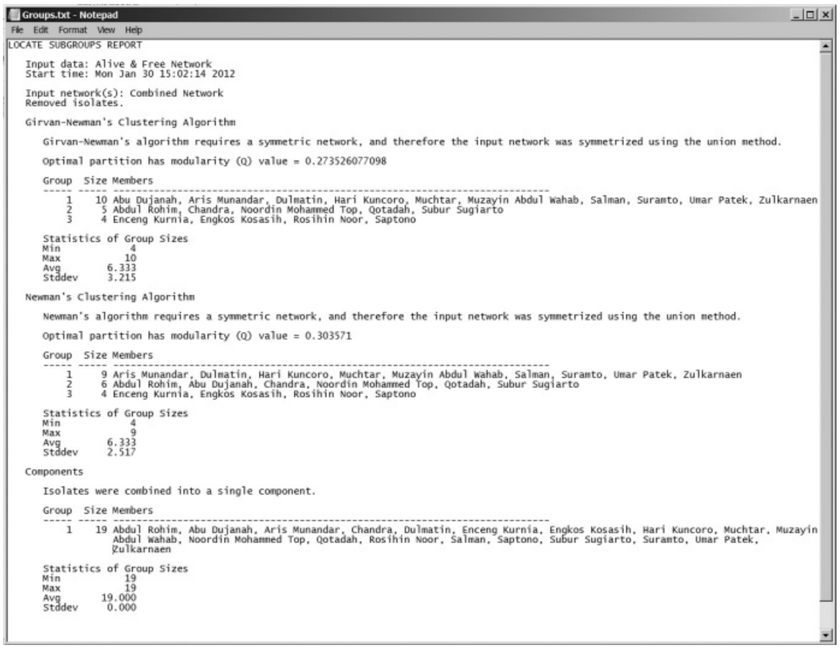
[✓ Automatically compute number of groups] Find this many groups: 3

Figure 6.31. ORA Locate Subgroups Dialog Box

want to compare different partitions, you can "force" ORA to detect a particular number of groups. Finally, under the components tab, you can tell ORA to group isolates into a single cluster. Clicking "Next" brings up another dialog box (not shown); accept ORA's defaults (although at some point you may want to explore what these options do), then click "Next" and the final dialog box appears (not shown). Here, tell ORA where (i.e., which folder) you want to save a text file of the report and under what name. Click "Finish" and a report will appear in the main screen's report panel as well as an HTML page in your computer's browser.

Figure 6.32 presents a portion of the text file report generated by ORA. It shows the results of the Girvan-Newman and the Clauset, Newman, Moore (just listed as Newman) algorithms. The results are almost identical except that the algorithms disagree where to place Abu Dujanah. They assign him to different groups. Because the Clauset, Newman, Moore algorithm has a slightly higher modularity score, we would generally be inclined to follow its results. Interestingly, the latter algorithm assigns Dujanah to the same group of individuals that faction analysis did. Nevertheless, the fact that these various algorithms disagree as to where to place Dujanah lends additional support to the possibility that he may be a broker in the network. Let's now turn to see how to detect Newman groups from ORA's visualizer.

After highlighting the combined network in the Meta-Network Manager, click on the "Visualize This Network" button in the Information/Editor panel. This will call up ORA's visualizer and only highlight

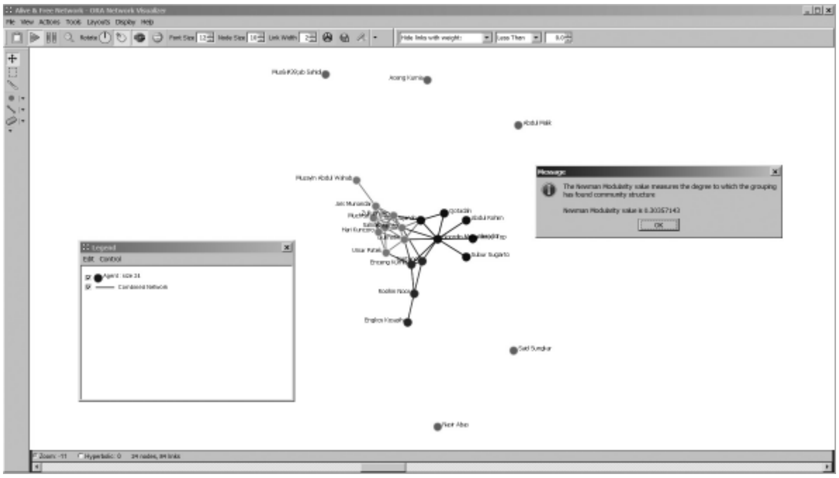


```
Groups.txt - Notepad
LOCATE SUBGROUPS REPORT

  Input data: Alive & Free Network
  Start time: Mon Jan 30 15:02:14 2012

  Input network(s): Combined Network
  Removed isolates.

  Girvan-Newman's Clustering Algorithm

    Girvan-Newman's algorithm requires a symmetric network, and therefore the input network was symmetrized using the union method.

    Optimal partition has modularity (Q) value = 0.273526077098

    Group  Size Members
    ----- ----- -----------------------------------------------------------------------------
        1    10 Abu Dujanah, Aris Munandar, Dulmatin, Hari Kuncoro, Muchtar, Muzayin Abdul Wahab, Salman, Suramto, Umar Patek, Zulkarnaen
        2     5 Abdul Rohim, Chandra, Noordin Mohammed Top, Qotadah, Subur Sugiarto
        3     4 Enceng Kurnia, Engkos Kosasih, Rosihin Noor, Saptono

    Statistics of Group Sizes
    Min           4
    Max          10
    Avg       6.333
    Stddev    3.215

  Newman's Clustering Algorithm

    Newman's algorithm requires a symmetric network, and therefore the input network was symmetrized using the union method.

    Optimal partition has modularity (Q) value = 0.303571

    Group  Size Members
    ----- ----- -----------------------------------------------------------------------------
        1     9 Aris Munandar, Dulmatin, Hari Kuncoro, Muchtar, Muzayin Abdul Wahab, Salman, Suramto, Umar Patek, Zulkarnaen
        2     6 Abdul Rohim, Abu Dujanah, Chandra, Noordin Mohammed Top, Qotadah, Subur Sugiarto
        3     4 Enceng Kurnia, Engkos Kosasih, Rosihin Noor, Saptono

    Statistics of Group Sizes
    Min           4
    Max           9
    Avg       6.333
    Stddev    2.517

  Components

    Isolates were combined into a single component.

    Group  Size Members
    ----- ----- -----------------------------------------------------------------------------
        1    19 Abdul Rohim, Abu Dujanah, Aris Munandar, Chandra, Dulmatin, Enceng Kurnia, Engkos Kosasih, Hari Kuncoro, Muchtar, Muzayin
                Abdul Wahab, Noordin Mohammed Top, Qotadah, Rosihin Noor, Salman, Saptono, Subur Sugiarto, Suramto, Umar Patek,
                Zulkarnaen

    Statistics of Group Sizes
    Min          19
    Max          19
    Avg      19.000
    Stddev    0.000
```

Figure 6.32. ORA Locate Subgroups Report (Text File)

the ties of the combined network. Next, issue the *Display>Node Appearance>Node Color>Color Nodes by Newman Grouping* command. This will color the actors by the Newman group to which they have been assigned as well as provide a modularity score (Figure 6.33). As you may have noticed, the visualizer has only one of the two

[ORA Visualizer] Display>Node Appearance >Node Color >Color Nodes by Newman Grouping



```
Message
The Newman Modularity value measures the degree to which the grouping has found community structure

Newman Modularity value is 0.30357143
```

Figure 6.33. ORA's Visualization of Newman Groups

Newman group algorithms (Clauset, Newman, Moore), and it does produce the same results (i.e., the modularity score is the same, and although it is difficult to distinguish the groups with ORA's grayscale option, the individuals have been assigned to the same groups). The ease with which you can identify groups in ORA's visualizer makes it an attractive option. However, if you want a report that you can print or incorporate into a report, then you probably will want to generate ORA's "Locate Subgroups" report.

6.6 Summary and Conclusion

In this chapter we have explored four sets of algorithms for detecting cohesive subgroups: components, cores, factions, and Newman groups. There are several more we have not considered. What should be clear by now is that we may have to use multiple algorithms before we succeed in detecting cohesive subgroups. Of course, when working with small groups, we can often detect them visually; however, as groups become larger it becomes increasingly necessary to turn to algorithms in order to identifying subgroups. Once they are detected, then we can possibly use this information for constructing strategies to disrupt the larger network, such as targeting amnesty programs at peripheral members or using disinformation campaigns to sow seeds of distrust between two or more subgroups. As social movement scholars have noted, infighting can led to a decline of an insurgency or movement (McAdam 1982, 1999).

What have we learned about Noordin's network, and how might this knowledge inform our attempts to disrupt it? Components analysis did not prove terribly useful although if later we are interested in focusing on a particular subsection of the network, then it could be a useful first step. By contrast, *k*-core analysis was somewhat more helpful. Our *k*-core analysis of the alive trust and operational networks identified a series of dense clusters of individuals. Because higher *k*-cores are nested within lower ones, *k*-core analysis can be useful for identifying clusters that are deeply embedded in particular networks and those that are not. Because research has shown that individuals deeply embedded in networks are unlikely to defect (Popielarz and McPherson 1995; Stark and Bainbridge 1980), a reconciliation campaign directed at Noordin's network will most likely do better if it targets individuals in the lower *k*-cores of the alive trust and operational networks. Indeed, because individuals who are currently incarcerated comprise a captive audience of sorts, it would make sense to identify those individuals who are both in jail and not deeply embedded in the network. They are probably the ones most open to leaving their violent pasts behind.

The use of factional analysis and Newman groups with the alive and free combined network yielded similar but not identical subgroups. In particular, three individuals were difficult to place – Dulmatin, Umar Patek, and Abu Dujanah – suggesting that they may be in positions of brokerage within the network. One strategy would be to remove them from the network, which would cause the network to fragment and possibly become less effective. However, as we have noted previously, this is not always the case. Networks often heal quickly, and the removal of key individuals can make them harder to track (Arquilla 2009:34). Another approach is to target individuals in positions of brokerage as portals for the diffusion of disinformation through a network (Anonymous 2009). In the case of Noordin's network, a deception campaign that used Dulmatin, Patek, and Dujanah to sow seeds of distrust between the subgroups could cause the network to implode or at least cease to function. A third approach recognizes that our information is often incomplete, and that rather than removing brokers and other key actors from the network, it is sometimes better to monitor them in the short run with the idea of improving our knowledge of the network and improving the selection of strategies adopted in the future (Arquilla 2009:34). Finally, this discussion of brokerage and key actors provides a nice segue into the following two chapters: Chapter 7, which looks at metrics that help to identify central, powerful, and prestigious actors in a network, and Chapter 8, which looks specifically algorithms used for identifying a network's brokers.