SNA Basics #10
Roles and Positions: Structural Equivalence

Positions and roles are important (and related) concepts in social theory.[1] A position (e.g., student) is typically connected to a particular role or set of roles (e.g., attending class, writing papers, studying). Moreover, such positions are typically located within a larger system of positions (e.g., fellow students, professors, staff, administrators), which is why social network theorists believe that individuals occupying a particular position/role will have a similar pattern of ties and exhibit similar patterns of behavior. Thus, they have developed a number of algorithms for identifying actors who (at least in theory) occupy equivalent positions within a network. Actors who do are said to be structurally or relationally equivalent. There are a number of different types of algorithms for identifying structural equivalent actors (e.g., structural equivalence, automorphic equivalence, regular equivalence) and multiple algorithms within each of these types (e.g., there are three regular equivalence algorithms implemented in UCINET).[2] In this lab, however, we only look at one of these sets of algorithms: structural equivalence. This isn't because the others aren't important. It's just that the method for computing automorphic and regular equivalence is nearly identical to structural, so there is really no point in illustrating the process. It would just make this lab unnecessarily long (it's already plenty long).

Two actors are considered structurally equivalent if they have identical ties with each other and all other actors in the network. Of course, seldom do two actors exhibit the exact pattern of ties, so we generally use a similarity threshold to identify which actors to consider structurally equivalent. Those that are, are then assigned to the same equivalence class or "block," which is why this approach is sometimes referred to as "blockmodeling." Blockmodel algorithms group actors into clusters and determine the relations between these clusters (e.g., one cluster of insurgents may be located in the center and another at the periphery). While on the surface this clustering technique may seem straightforward, there are numerous blockmodeling algorithms for identifying structurally equivalent actors. Some algorithms are computationally intense, which limits their application to relatively small networks.

Part I – Structural Equivalence in UCINET (Sample Data)

[UCINET] Data>Display

1. We begin by examining a relatively small network that Wasserman and Faust use because it is easier to illustrate certain blockmodeling techniques with smaller datasets than with larger ones.[3] In UCINET locate and display the Wasserman and Faust structural equivalence dataset (`WFSE.##h`), using the *Data>Display* command. It should look similar to Figure 1 (next page – a network graph of the dataset appears to the right of the matrix).

[1] Wouter de Nooy , Andrej Mrvar, and Vladimir Batagelj. 2011. *Exploratory Social Network Analysis with Pajek*. Cambridge, UK: Cambridge University Press.
[2] While can be a bit confusing, the theoretical concept of structural equivalence should not be confused with the algorithm of the same name that is used to identify structurally equivalent actors. All equivalence algorithms are designed to identify structurally equivalent actors. It just so happens that one set of these algorithms goes by the name, "structural equivalence."
[3] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications.* Cambridge, UK: Cambridge University Press: 364.

SNA Basics #10
Roles and Positions: Structural Equivalence

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	0	1
2	0	0	0	0	1	0	1	0	0
3	0	1	0	0	1	1	1	1	0
4	1	0	0	0	0	0	0	0	1
5	0	1	0	0	0	0	1	0	0
6	0	1	1	0	1	0	1	1	0
7	0	1	0	0	1	0	0	0	0
8	0	1	1	0	1	1	1	0	0
9	1	0	0	1	0	0	0	0	0

Figure 1: Wasserman and Faust (1994) Example Dataset

2. Euclidean distance is a common approach for identifying structurally equivalent actors. Euclidean distance algorithms take network data and calculate a set of points in *n*-dimensional space, such that the distances between them correspond as closely as possible to the input proximities. Euclidean distance differs from distance in graph theory. The latter measures the distance between two actors in terms of the number of lines in the path that connects the two points (i.e., path distance), while the former measures the distance between two actors in terms of the most direct route between them (i.e., as the crow flies).[4]

Network>Roles & Positions>Structural >Profile

3. UCINET's Euclidean distance structural equivalence algorithm can be accessed with the *Network>Roles & Positions>Structural>Profile* command, which brings up a dialog box similar to Figure 2. Here accept most of UCINET's default settings, including Euclidean distance default.[5] The one exception is that the "For binary data…" pull-down menu should be set to "Yes" not "No." Also, be sure that the Wasserman and Faust file is indicated as the input dataset (`WFSE`). Click OK. UCINET should create two new files: a structural equivalence matrix (`SE`) of Euclidean distances between actors and a structural equivalence partition (`SEPart`) that assigns each actor to its respective equivalence class.

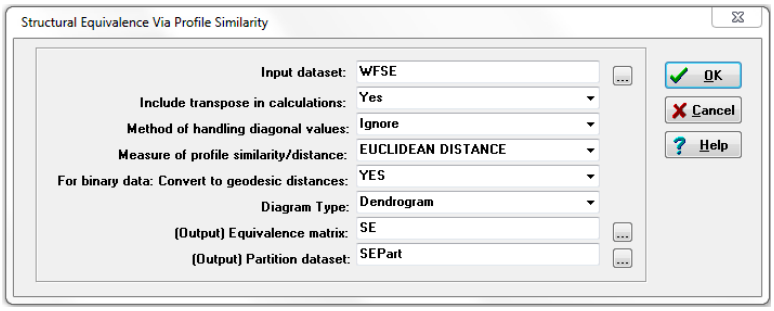

Figure 2: Structural Equivalence Profile Similarity Dialog Box

[4] John Scott. 2000. *Social Network Analysis: A Handbook*. Thousand Oaks, CA: Sage Publications: 157.
[5] The "Measure of profile similarity/distance" pull-down menu offers an array of methods for estimating structural equivalence, and if you click on UCINET's help button, you will find a brief description of the various methods.

Data>Display

4. Display the structural equivalence matrix of Euclidean distances between actors (Figure 3). The smaller a number in a particular cell, the more similar those two actors are. A score of 0.00 indicates perfect similarity, which means that the two actors share an identical pattern of ties. Looking at the matrix, you can see that actor 2 is the same distance away from actors 3 & 8 (16.00) and actors 1, 4 & 9 (26.533) and perfectly similar to actors 5 & 7 (0.00).

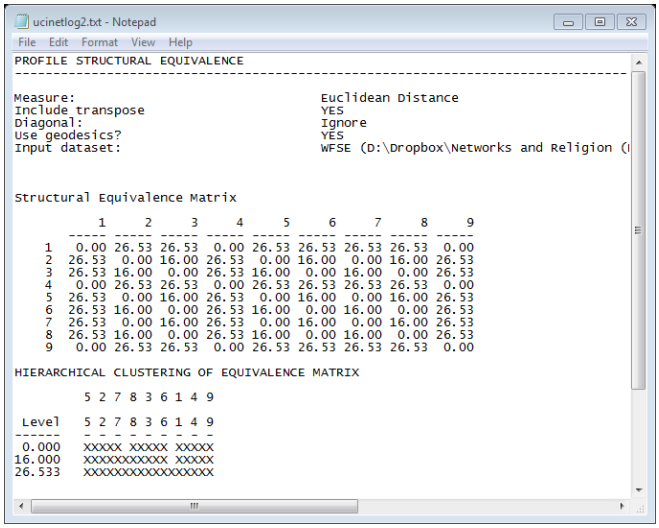

Figure 2: Structural Equivalence Matrix and Hierarchical Clustering

5. The next step in the blockmodel process is to permute (i.e., reorder the rows and columns) and partition the original matrix so that actors who are assigned to the same block occupy adjacent rows and columns. To do this we use both the original matrix and the structural equivalence partition file (`SEPart`). Let's begin

Data>Display

by displaying the partition file, which should look similar to Figure 3 (next page). Note that the file actually includes three partitions (columns): one at similarity level 0.00 (the highest level of similarity), one at 16.00, and one at 26.53 (the lowest level of similarity). These correspond to the rows in the "hierarchical clustering of equivalence matrix" displayed in Figure 2 above. If we want to partition the network in structural equivalence classes at the highest level of similarity between actors, then we would want to use the first partition (column). Similarly, if we want to partition the network at the lowest level of similarity, then we would use the third partition (column). Note, however, that at the lowest level of similarity, all of the actors are assigned to the same equivalence class, so it isn't of much use. If we were to use the second partition (column), the actors would be assigned to two classes. Here, we will use the first partition (column), which will assign the actors to three equivalence classes.

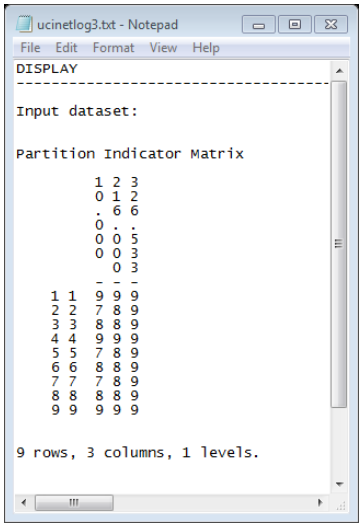

Figure 3: Structural Equivalence Partition

Transform >Aggregate (include CSS) >Block

6. To do this we use the *Transform>Aggregate (include CSS)>Block* command to permute the original matrix and generate an image matrix. In the resulting dialog box you need to indicate what the input dataset is and the row/column partition file is. As you can see in the following dialog box, I instructed UCINET to use the first partition (Column 1) of the "SEPart" partition file.

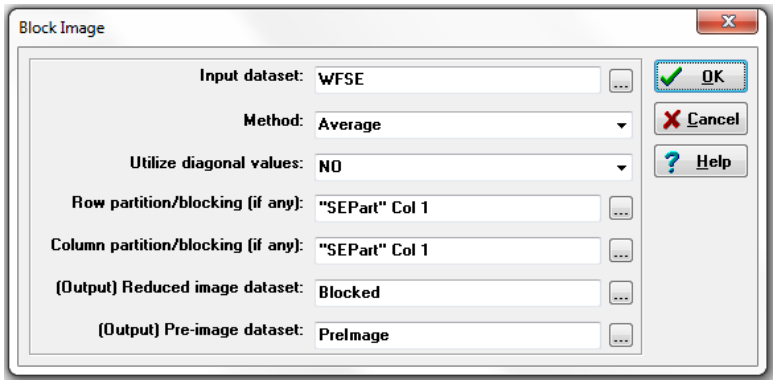

Figure 4: Block Image Dialog Box

7. The resulting output file provides three pieces of information. First, it tells you which blocks the actors have been assigned to (not shown). Next, it provides a permuted and partitioned matrix (Figure 5) where actors in the same equivalence class are grouped together and partitioned from the other classes. As you can see from the figure, actors 5, 2 and 7 have been assigned to the first block or class, 8, 3 and 6 have been assigned to the second block, and 1, 4 and 9 have been assigned to the third. You can also see from the permuted matrix that actors within in the first and third blocks only send ties to other actors within their own block, whereas the actors within the second block send ties to each other and members of the first block. This pattern of where blocks send ties can be captured in two ways: with an image matrix and a graphical representation of the image

matrix. The image matrix (i.e., the reduced block matrix) is the last piece of information that the output file provides (Figure 6).

```
    5 2 7   8 3 6   1 4 9
   ----------------------
5 |   1 1 |       |       |
2 | 1   1 |       |       |
7 | 1 1   |       |       |
  ------------------------
8 | 1 1 1 |   1 1 |       |
3 | 1 1 1 | 1   1 |       |
6 | 1 1 1 | 1 1   |       |
  ------------------------
1 |       |       |   1 1 |
4 |       |       | 1   1 |
9 |       |       | 1 1   |
   ----------------------
```

Figure 5: Permuted and Partitioned Matrix

8. The image matrix (Figure 6) collapses each block/class of the permuted matrix into a single cell where the number appearing in the cell indicates the density of ties between the actors of that block. In this case in each of the blocks all possible ties are present, so the density within each block is 1.00. Since all of the actors in the second block (i.e., 8, 3 & 6) have ties to all of the actors in the first block (i.e., 5, 2 & 7) the density of that cell is 1.00 as well. However, because ties are completely absent between the other blocks, their density is 0.00. The image matrix's corresponding graph (i.e., the reduced graph) captures the relationship between blocks. Although it is quite small here, if you look closely you can see that block two is the only block that sends ties to any other block (block one), while all three blocks send have reflexive ties (loops); that is, members send ties to other members of the block.

```
Reduced BlockMatrix

          1     2     3
      ----- ----- -----
  1   1.000 0.000 0.000
  2   1.000 1.000 0.000
  3   0.000 0.000 1.000
```

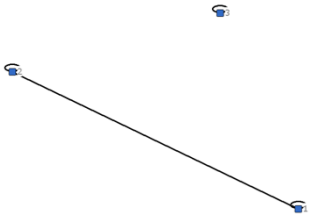

Figure 6: Image Matrix and Corresponding Graph

9. As you may have guessed, the density within and between blocks seldom exactly equals either 1.00 or 0.00 (in other words, actors are seldom perfectly similar or dissimilar), so we generally have to choose some sort of criterion that distinguishes "one-blocks" (a.k.a., complete blocks) from "zero-blocks" (a.k.a., null blocks). We will take up that topic in the next section where we examine the Anabaptist Leadership network.

Part II – Structural Equivalence in UCINET (Anabaptist Leadership)

1. Since there is no point of examining the Anabaptist Leadership network using the Euclidean algorithm since we would simply be repeating the process illustrated above, we'll move on to another algorithm: CONCOR, which was one of the earliest approaches to identifying structurally equivalent actors. CONCOR, which stands for "CONvergence of iterated CORrelations," is based on the fact that repeated calculation of correlations between a matrix's rows (or columns) eventually results in correlation matrix consisting of only +1.00's and -1.00's.[6] It begins by correlating each pair of actors, and then each row of the resulting actor-by-actor correlation matrix is then extracted and correlated with each other row. The algorithm repeats this process over and over again until all of the coefficients approach either +1.00 or -1.00. It then splits the data into two sets (i.e., the +1.00 set and the -1.00 set), and then it repeats the process for each of the sets (or at least those sets with two or more actors) until it runs out of actors to split or arrives at the number of "splits" indicated by analysts at the outset.

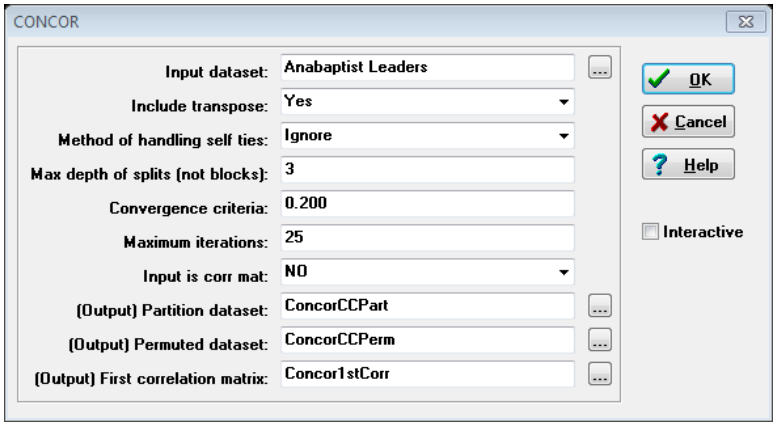

Figure 7: CONCOR Dialog Box

Network>Roles & Positions>Structural >CONCOR>Standard

2. The CONCOR command in UCINET can be found under the Network menu. UCINET now includes two CONCOR algorithms: (1) the standard algorithm where we simply indicate the number of splits we want CONCOR to estimate and (2) an interactive one, which provides a bit more control over how many and where we want the splits to occur. We will begin with the standard algorithm, which brings up a dialog box that should look similar to the one above (Figure 7). Note that I've loaded the binary (i.e., dichotomized rather than valued) trust network as the input dataset and indicated that I want CONCOR to split the data a maximum of 3 times, which should give us no more than six blocks of structurally equivalent actors.

[6] Wasserman and Faust, p. 376.

3. UCINET generates a dialog box (not shown – it may be hidden behind the output file) asking you if you want to see a dendrogram, which graphically illustrates the splits in the data that have occurred). Click OK in order to see this. The output file (not shown) consists of three panels: the first-order correlation matrix (as explained above), a partition diagram, and a density matrix. The partition diagram indicates where the splits in the data occurred. The bottom row of "x's" indicates the two groups that the first split identified, while the top row indicates the four groups that the second split identified. If I had asked CONCOR to split the data three times, there would be three rows of x's in the partition diagram. Listed above the rows of x's are the names of the actors, indicating the block to which they have been assigned. UCINET also generates a partition matrix/file that we will use to generate the block model as we did above.

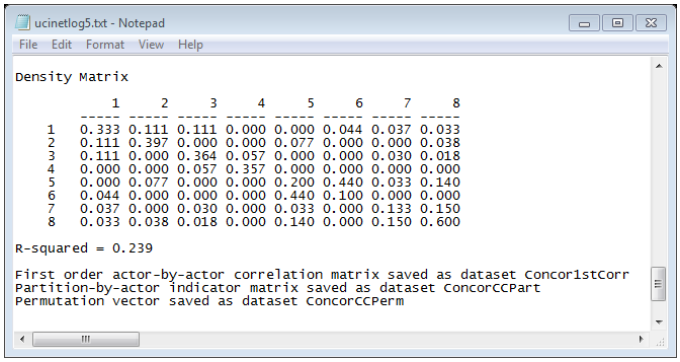

Figure 8: CONCOR Density Matrix

4. As before the density (image) matrix (Figure 8) found at the bottom of the output collapses each block/class of the permuted matrix into a single cell where the number appearing in the cell indicates the density of ties between the actors of that block. Note that none of the densities equal 1.00 and about half equal 0.00, so it isn't as straightforward as it was above to create an image matrix. We take up how in the next step. Note also that UCINET provides a measure of fit (R-squared = 0.239) that compares partitioned data matrix with an ideal structure matrix (i.e., where cell densities equal block means). In this case, the higher the R-squared the better the fit.

5. So, how do we identify which blocks to classify as one/complete blocks or zero/null blocks? There are a number of different approaches. The zero block (or lean fit) approach only classifies a block as a null block if its density equals 0.00 (i.e., there's a complete absence of ties).[7] All the rest are classified as one (complete) blocks. If we apply this approach to our current network, the final image matrix (or blockmodel) looks like Figure 9. This tells us that half of the blocks are sending and receiving ties and half are not. Put differently, half of the

[7] Wasserman and Faust, p. 399.

blocks are complete and half are null. If we used the one block approach,[8] which only classifies a block as complete if its density equals 1.00, we get a final image matrix of all 0s and no 1s, which in this case (and in most cases) is unhelpful.

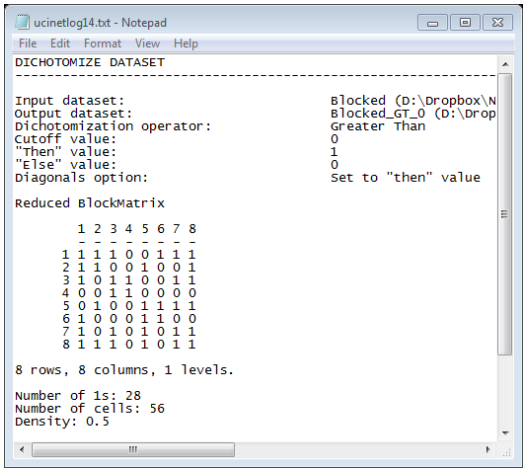

Figure 9: Final Image Matrix (Blockmodel) using Zero Block Method

6. Probably the most common approach for distinguishing complete blocks from null blocks is to set a threshold such that if a particular block's density is greater than or equal to that threshold, it is classified as a complete block, and if it is not, it is classified as a null block.[9] The most common threshold is the density of the overall network (although some have used thresholds that vary across blocks),[10] which if you recall, can be obtained using UCINET's *Network>Cohesion>Density>Density Overall* command.

Network>Cohesion>Density>Density Overall

7. The density of Anabaptist Leader network is .083, so to create a final image matrix using this as a threshold, we turn again to the *Transform>Aggregate (include CCS)>Block* command to first generate an image matrix. As before, we indicate the input dataset and the partition file (here we use column 1) and click OK. Once again UCINET's output (not shown) provides us with a summary of the blocks to which each actor is assigned, the permuted network, and a reduced block matrix (which should look identical to the one in Figure 8).

Transform>Aggregate (include CSS)>Block

[8] Ibid, p. 400.
[9] Ibid.
[10] Ibid, p. 400-401.

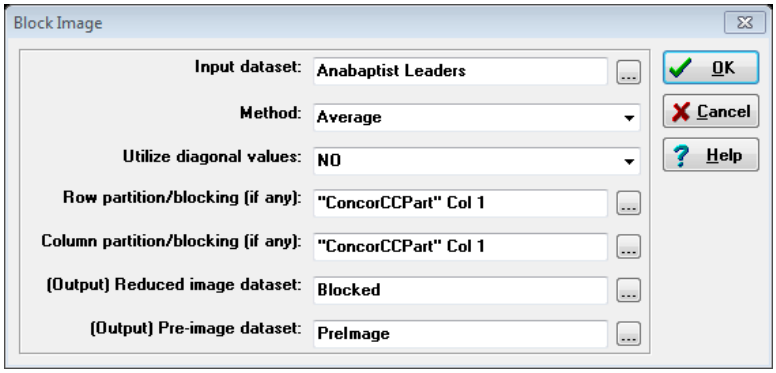

Figure 10: Block Image Dialog Box

8. The final step in the process involves dichotomizing (binarizing) the network so that cells with a density of .083 or greater are classified as complete blocks, while all the rest are classified as zero blocks. We do this with the *Transform>Dichotomize* command, which brings up the following dialog box (Figure 11).

Transform>Dichotomize

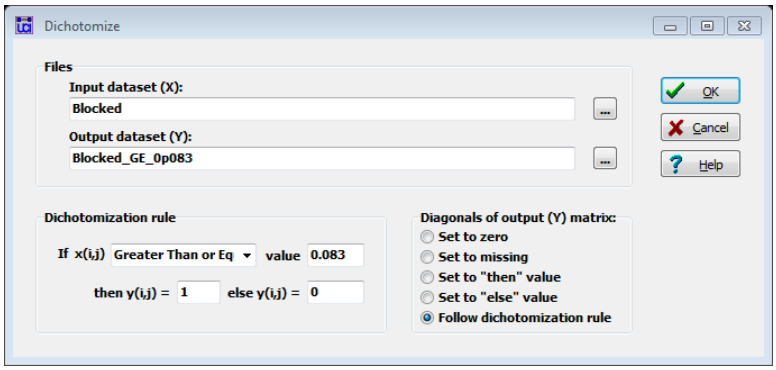

Figure 11: Dichotomize Dialog Box

9. Note that I have set the cut-off operator to be "greater than or equal" and the cut-off value at 0.083. Also, I have told UCINET that the diagonal should follow the dichotomization rule and not be set to zero. When you click OK, UCINET's output will provide you with a final image matrix (Figure 12) that can be analyzed as a matrix or a network map. The output does provide us with some useful information. For instance, it tells us that 10 out of the 56 cells are populated with "1s", which is 17.857% of the cells. We can also see that the first three blocks appear to interact with one another, as do the last four. However, the fourth only interacts with itself (see Figure 13). It is probably worthwhile exploring the membership of these various blocks. UCINET provided us when we used the *Transform>Aggregate (include CCS)>Block* command, so we may want to go back and issue the command again. However, we can identify block membership when we use the interactive version of CONCOR, which is where we turn to next.

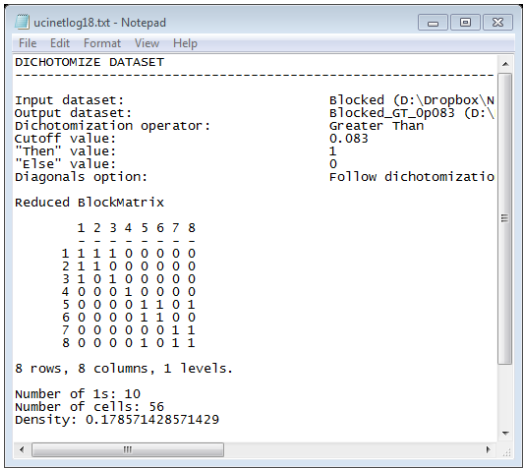

Figure 12: Final Image Matrix (CONCOR)

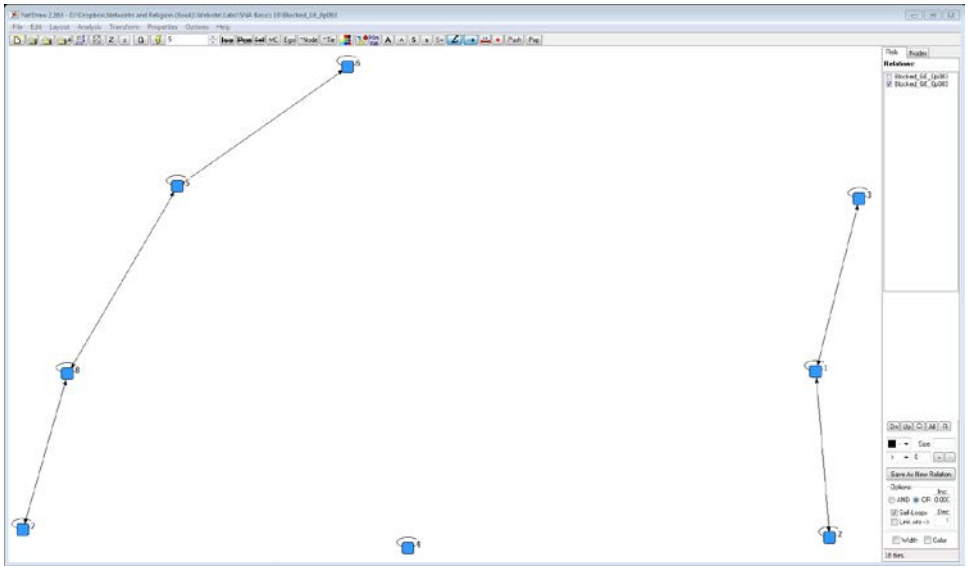

Figure 13: Graph of Final Image Matrix (CONCOR)

Network>Roles & Positions>Structural >CONCOR>Interactive

10. The interactive CONCOR algorithm provides analysts with a little more flexibility in determining how often and where we want to split the network. You access it (Figure 14) using the *Network>Roles & Positions>Structural> CONCOR>Interactive* command. Once you identify the data you intend to analyze, you have to click on the "Load" button before you can work with the data. You begin by selecting a block of nodes and clicking the "Split" button, which will split the data into two. From that point on you can choose where you want the splits to occur. Here, I simply replicated what the standard algorithm did, but I could choose to split these further. Clicking on the "Densities" button allowed me to pull up a window that indicates the corresponding density matrix as well as how well the data fit (R-squared = .235), which is approximately the same as above. However, additional splits yield a better fit (not shown). Although it's a little hard to see, I've selected the fourth block (0.1.2.2), and the log tells us

that it consists of 8 actors and provides us with their names. Once we're satisfied with how the data are clustered, we simply click "Save" (close the density matrix window first), which will generate a partition that we can use to create a final image matrix as we did before.

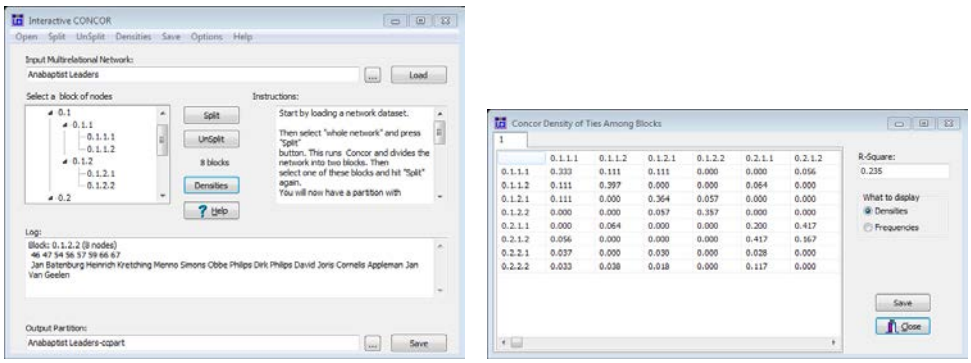

Figure 14: Interactive CONCOR Dialog Box

Network>Roles & Positions>Structural >Optimization>Binary

11. We'll consider one final algorithm available in UCINET that shares a lot of similarities to the primary one implemented in Pajek, which is why we have saved it for last. We access it with UCINET's *Network>Roles & Positions> Structural>Optimization>Binary* command. Note that this approach provides you with an option of using either valued or binary data. We'll continue to use binary data here so that the results are comparable to the results we got above.

12. This approach permutes (rearranges) the rows and columns of the matrix, trying to find the best fit for the number of blocks the analyst designates at the outset (in this case, eight). Fit is determined by an error score (the lower the better) that compares the final model with a perfect model. Recall that in the case of perfect structural equivalence, in a complete block all possible ties are present, while in a null block no ties are present. Consequently, with the optimization approach, an error is considered to occur when a tie is present in a null block or a tie is missing in a complete block. The optimization approach continues until it can no longer lower the error score. That said, most of the time increasing the "number of blocks" reduces the error score, so it's probably best to begin with a "theory" as to the number of blocks, rather than continually increasing the number until the error score stops decreasing. Here (Figure 15), I use eight (8) blocks for illustrative purposes, but again it is best that theoretical concerns drive our analysis.

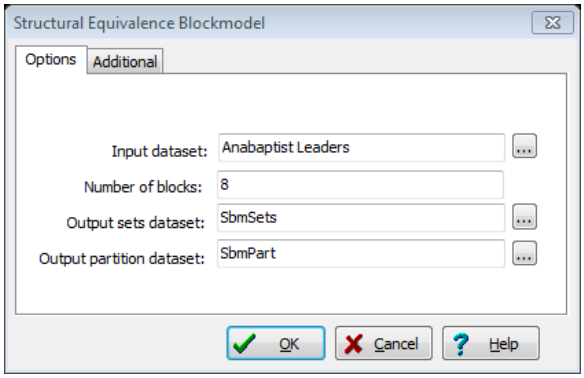

Figure 15: Optimization Dialog Box

13. When you click OK, UCINET generates a report (not shown) that indicates the final (and beginning) error score as well as an R-squared fit statistic. As the reports generated by the other equivalence algorithms did, this one indicates the blocks to which each actor is assigned, provides a permuted and partitioned (blocked) adjacency matrix, and displays a density table. In addition to the report, the algorithm generates a partition that allows analysts to create a final image matrix (Figure 16) using the same steps we used above. The result suggests a different result from our previous analyses (see Figure 17). This shouldn't be too surprising since the algorithms we use vary in their assumptions. Nevertheless, it serves as a useful reminder that algorithms vary in their assumptions and to rely on them blindly is probably not a great strategy.

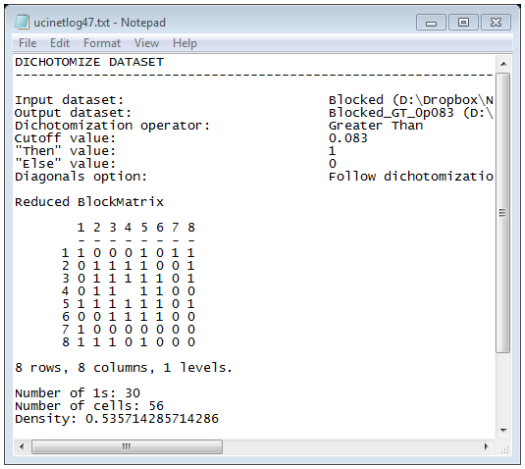

Figure 16: Final Image Matrix (Optimization)

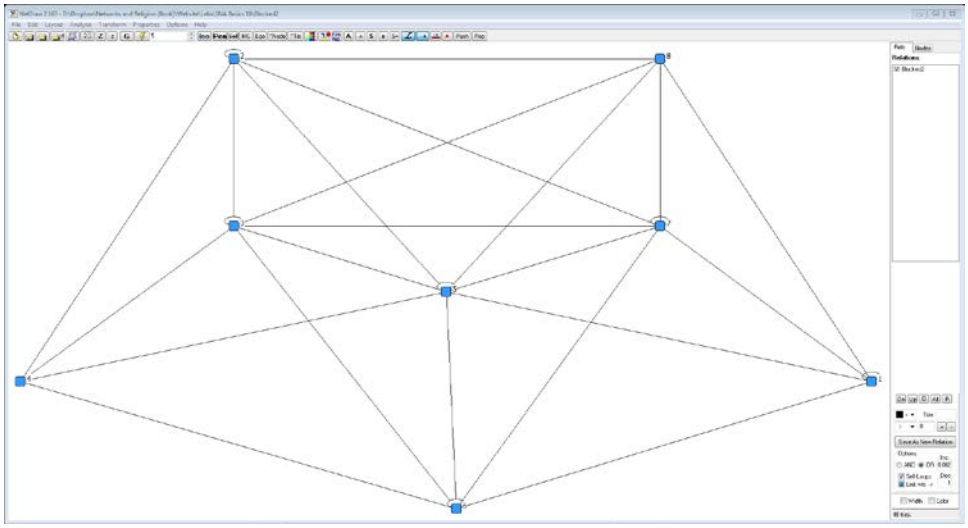

Figure 17: Graph of Final Image Matrix (Optimization)

14. On a final note, we can also visualize the original network where the nodes are colored by their block assignment in NetDraw, using the partitions created by CONCOR and optimization algorithms. Figure 18 is a visualization of the network where the colors of the nodes reflect the blocks identified by the CONCOR algorithm, while Figure 19 visualizes the same network using the optimization partition.

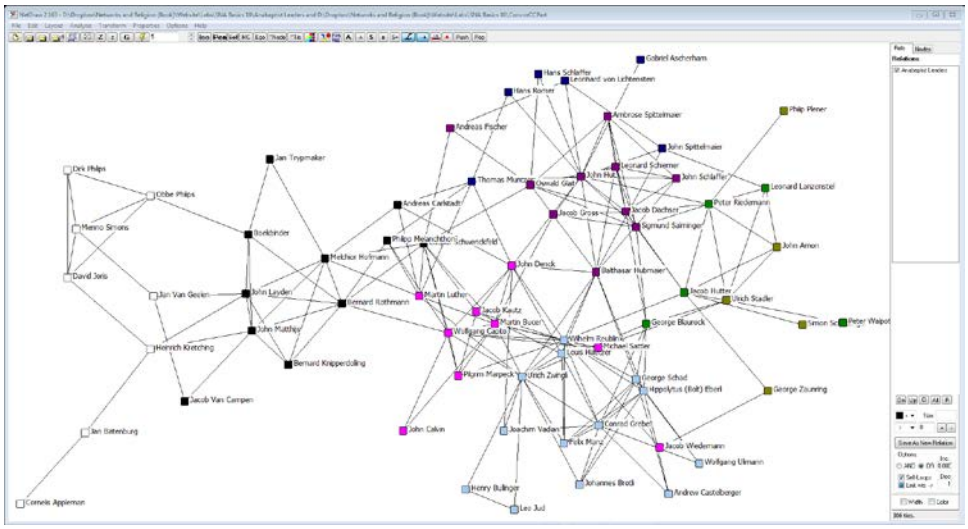

Figure 18: Anabaptist Leader Network (with CONCOR Partition)

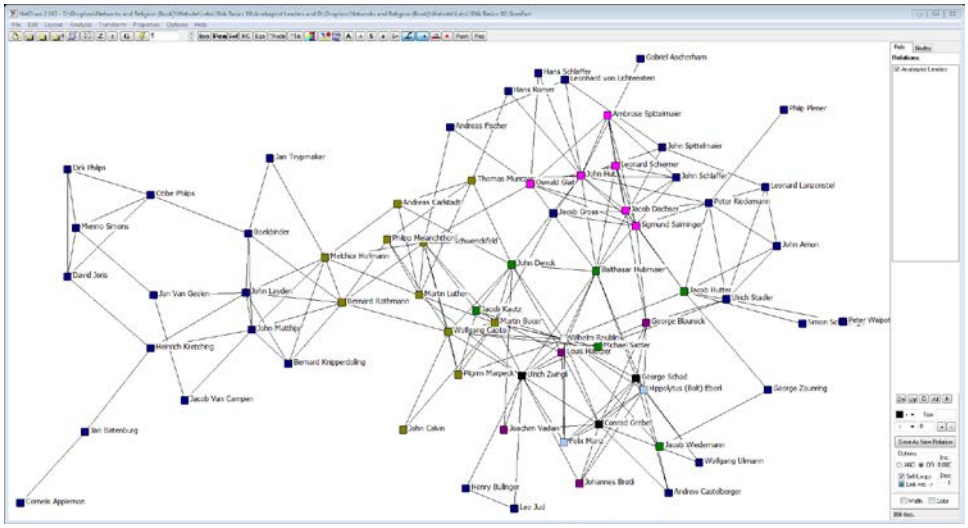

Figure 19: Anabaptist Leader Network (with Optimization Partition)

Part III – Roles, Positions and Structural Equivalence in Pajek

1. The three approaches to structural equivalence illustrated above do not exhaust those that are available in UCINET. Hierarchical clustering is another common approach implemented in UCINET, one that is also implemented in Pajek. Here, however, we focus on Pajek's optimization algorithm because it is the one for which Pajek (and its developers) are best known.[11]

[Pajek]
File>Network>Read

Network>Create Partition
>Blockmodeling
>Random Start

2. Begin by loading the trust network file (`Anabaptist Leaders.net`) into Pajek, using Pajek's *File>Network>Read* command. Next, use Pajek's blockmodeling command (*Network>Create Partition>Blockmodeling>Random Start*) to call up its blockmodeling dialog box (Figure 20, next page – you may need to uncheck the "Restricted Options" option). A pull-down menu allows analysts to indicate whether they want to generate structural or regular equivalence or define one themselves (Patrick Doreian, Vladimir Batagelj and Anuska Ferligoj have developed an approach to blockmodeling known as "Generalized Blockmodeling" that allows users to combine a variety of equivalence measures into a single model – see footnote 11). Analysts can also indicate how many iterations they want Pajek to perform in its search for the blockmodel partitions (yes, there can be more than one) that minimize the error score as well as how many clusters/blocks they want Pajek to identify. You can also indicate how many clusters (i.e., blocks) you want (here, I've chosen 8).

[11] See Wouter de Nooy , Andrej Mrvar, and Vladimir Batagelj. 2005. *Exploratory Social Network Analysis with Pajek*. Cambridge, UK: Cambridge University Press, pp. 273-291 and Patrick Doreian, Vladimir Batagelj, and Anuska Ferligoj. 2005. *Generalized Blockmodeling*. Cambridge: Cambridge University Press.

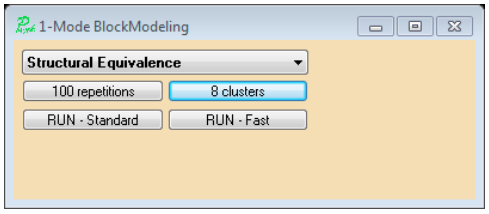

Figure 20: Pajek Blockmodeling Dialog Box

3. Click "RUN - Standard" and Pajek will generate a report that presents a final image matrix, a final error matrix, and a final error score (Figure 21). Note that Pajek's error score is lower than the one found by UCINET. We might be able to reduce the error score in UCINET by increasing the number of repetitions (iterations). UCINET's default is 50 (Pajek's is 100), and we can increase it by clicking on the "Additional" tab of the optimization dialog box (see Figure 15 above).

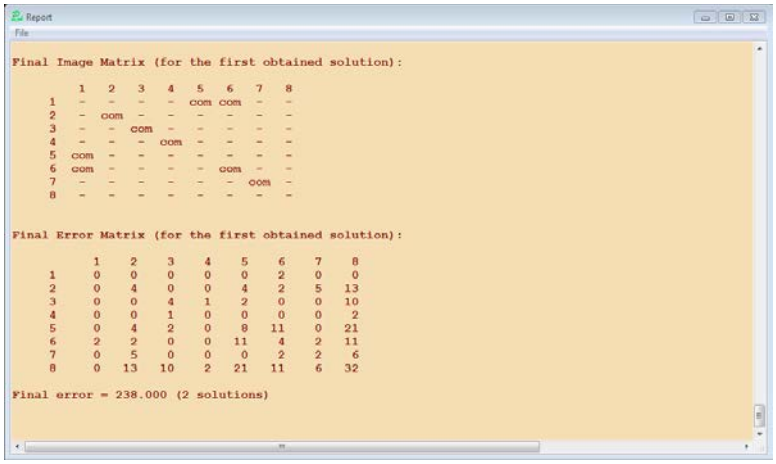

Figure 21: Pajek Blockmodeling Report

4. Pajek also allows you to generate a permuted matrix that may be helpful to inspect before drawing conclusions about a particular network's structure. To do this, make sure that a partition previously generated by the blockmodeling algorithms is highlighted in the top partition drop down box. Not just any partition, but the partition that you are analyzing. Then, create a permutation based on that partition, using Pajek's *Partition>Make Permutation* command. Next, select the *File>Network>Export as Matrix to>Options* command and check the "Use Partition for Vertex Labels" and "Thick Boundary Line" options. Finally, use the *File>Network>Export as Matrix to>EPS>Using Permutation + Partition* command to export the permuted matrix as an EPS file, which we can then (at least in theory) insert into a Word document (see Figure 22).[12]

Partition >Make Permutation

File>Network>Export as Matrix to>Options

File>Network>Export as Matrix to>EPS >Using Permutation + Partition

[12] I wrote "at least in theory" because inserting EPS files into Word documents on a PC has become difficult. However, you can still do it on a Mac. Pajek just added the feature of

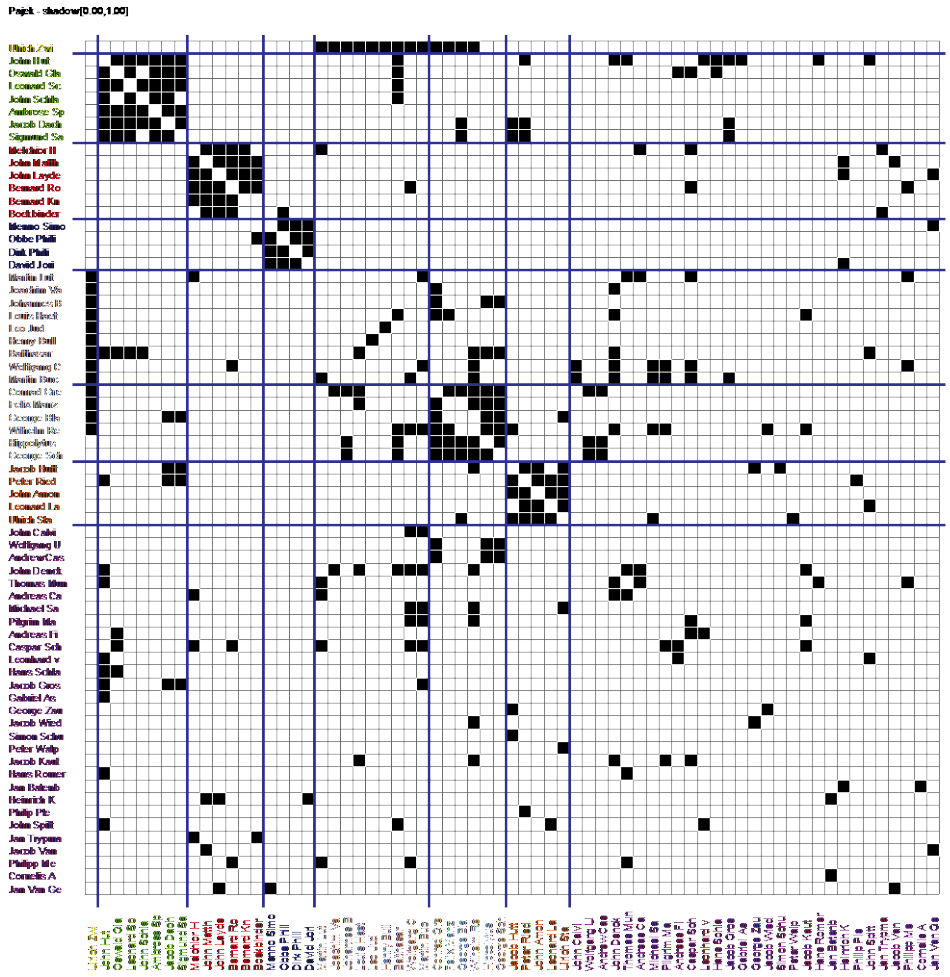

Figure 22: Permuted Matrix

5. A glance at Figure 22 tells us that members of the 2nd, 3rd, 4th, 6th, and 7th blocks interact primarily amongst themselves, while the sole member of the 1st block interacts with all the members of the 5th block and about half of the members of the 6th block. It's also clear that the 8th block isn't really a block at all. The only thing its members seem to have in common is that their interaction patterns are quite dissimilar from one another.

Draw>Network + Partition

6. Finally, we can draw the original network with the nodes colored by the block into which they've been partitioned using Pajek's *Draw>Network + Partition* command (Figure 23). Pajek may also find more than one solution (as it did here). If so, then all of the solutions are stored as partitions, which means that you can compare them in Pajek's draw screen (see Figure 24 for the second solution).

---

allowing you to export the permuted matrix in SVG and PDF format, but you need to install the program Inkscape first.

SNA Basics #10
Roles and Positions: Structural Equivalence

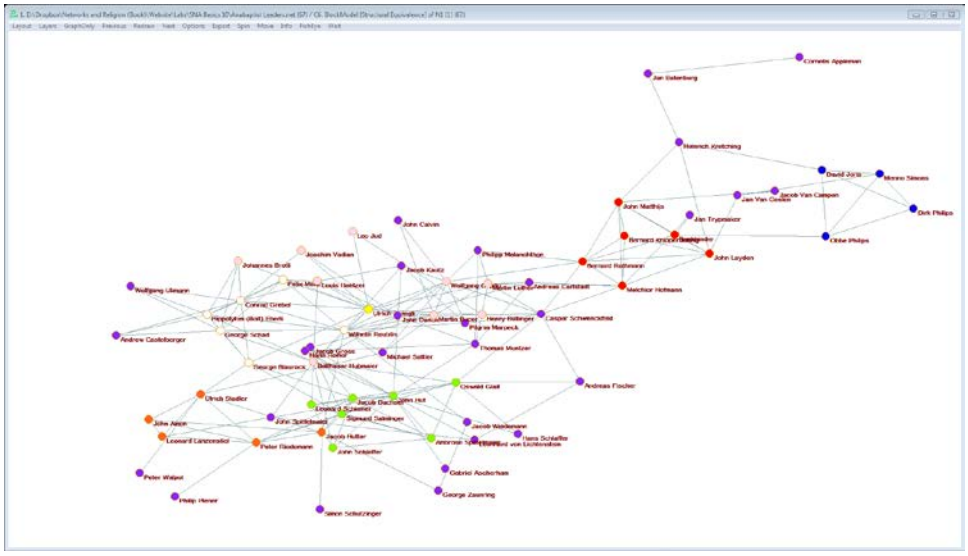

Figure 23: Anabaptist Leader Network Colored by Block (First Solution)

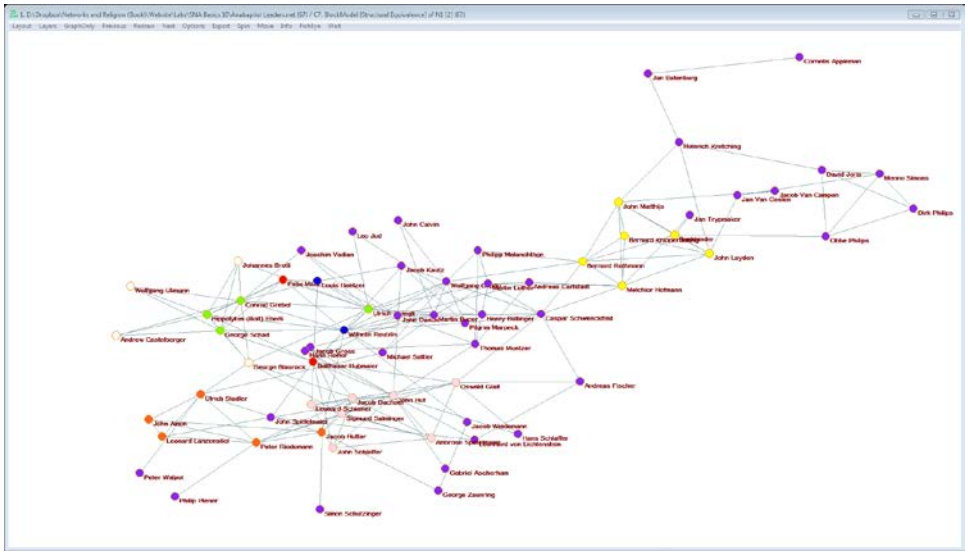

Figure 24: Anabaptist Leader Network Colored by Block (Second Solution)