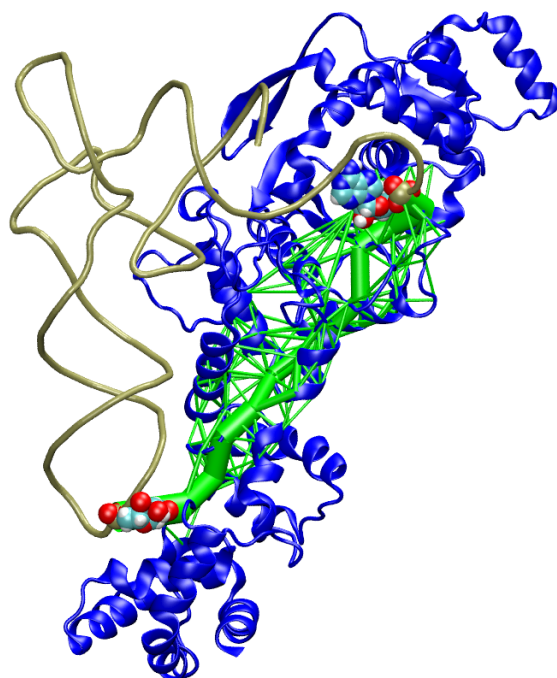


University of Illinois at Urbana-Champaign
Luthey-Schulten Group
NIH Resource for Macromolecular Modeling and Bioinformatics

Dynamical Network Analysis



VMD Developer: John Stone

Network Tool Developers Tutorial Authors

John Eargle
Anurag Sethi

John Eargle
Li Li
Zan Luthey-Schulten

July 6, 2012

A current version of this tutorial is available at
<http://www.scs.illinois.edu/schulten/tutorials/network/>

Contents

1	Introduction	3
1.1	Dynamical Networks for Protein and Nucleic Acids	3
1.2	Aminoacyl-tRNA Synthetases: Role in translation	4
1.3	Assembling the Tools for Network Analysis	4
1.3.1	Requirements	4
1.3.2	PATH Setup for Carma and CatDCD	6
1.3.3	Move gncommunities and subopt into the common Directory	7
1.4	Network File Preparation	7
2	Dynamical Network Representations	10
2.1	Load a network into NetworkView	10
2.2	Deactivate and activate subnetworks	11
2.3	Color the network	11
2.4	Weight the network using correlation data	12
3	Communities	13
3.1	Run gncommunities to generate network communities	13
3.2	Load community data files	13
3.3	Activate and color by community	13
3.4	Examine critical nodes and edges between communities	15
4	Optimal and Suboptimal Paths	15
4.1	Run subopt to obtain suboptimal paths	15
4.2	Load suboptimal path files	16
4.3	Activate and color the suboptimal paths	17
4.4	Load suboptimal path count into value	17
5	Application Programming Interface (API)	17
5.1	Beyond the Graphical User Interface (GUI)	17
5.2	getInterfaceEdges	18
5.3	getEdgeInfo	18
5.4	getEdgesByMetric	19
6	Acknowledgements	19

1 Introduction

1.1 Dynamical Networks for Protein and Nucleic Acids

Network models of various phenomena have become increasingly common in recent years. From transportation routes and the world wide web to systems biology and ecosystem modeling, network theory combined with increasing computational resources has provided useful strategies for thinking about and analyzing large data sets. The simplest networks consist of sets of nodes and edges that connect pairs of nodes. For example, in protein-protein interaction networks, nodes represent individual proteins, and if two proteins interact with one another, an edge is drawn between their nodes. In this tutorial we use concepts from network theory to describe and investigate residue-residue interactions within biomolecular systems.

The biological system we will be analyzing is the complex formed by glutamyl-tRNA synthetase (GluRS), tRNA^{Glu}, and glutamyl adenylate. This complex is responsible for recognizing and charging the tRNA with its cognate amino acid glutamic acid and was previously studied with the network methods presented in this tutorial [1, 2]. Interactions between the enzyme GluRS, tRNA, and substrate have been used to identify signaling pathways for recognition and to suggest a plausible reaction mechanism for the charging of tRNA. Different types of network analyses have been applied to protein structures in the past several years. For example, contact path length (CPL) is the mean shortest distance between all pairs of nodes in a network, and it is used as a measure of the size (sometimes called diameter) of a network. Conserved residues that greatly affect the CPL upon removal have been hypothesized to be important for allosteric signal transmission [3]. Snapshots from a short simulation of a modeled MetRS-tRNA complex indicated that the shortest path between protein residues interacting with the anticodon and the adenylate binding site was sensitive to conformational changes in the protein [4], but the tRNA and contacts with other identity elements on the tRNA were neglected in their study of the signal transmission. While the shortest path analysis identifies several nodes, the contribution of these nodes to communication in protein networks has not been examined, with few exceptions [5].

If there are multiple communication paths nearly equal in length, then not all residues along these paths need be considered as important for allostery. Instead, only residues or interactions that occur in the highest number of suboptimal pathways need to be conserved to guarantee an effective pathway for allosteric communication in the complex. In this work, we analyze entire protein-tRNA networks “weighted” by correlation data from long (20 ns) molecular dynamics (MD) simulations of the aaRS-tRNA complex in two functional states: before and after tRNA aminoacylation. The correlation, C_{ij} , in motion between nodes i and j defines information transfer between the nodes because motion of monomer (residue or nucleotide) i can be used to predict the direction of motion of monomer j . For both states, we determine the shortest path for communication along with the ensemble of suboptimal paths from all identity elements on

the tRNA to the active site of the synthetase.

The time averaged connectivity of the nodes is used to identify the substructure or communities in the network. The optimal community distribution is calculated using the Girvan-Newman algorithm [6], which has no free parameters, in contrast to other approaches [5, 7]. The community description allows us to compare the topology and modularity of networks for protein-RNA complexes. The conserved monomers involved in communication between communities are the critical nodes for communication within the network and are shown to occur in a majority of the suboptimal paths between the identity elements and the site of amino acid transfer at the 3' end of the tRNA.

1.2 Aminoacyl-tRNA Synthetases: Role in translation

Before beginning the actual tutorial, a small amount of background information on the cellular translation system may be helpful. The aminoacyl-tRNA synthetases (aaRSs) are key proteins involved in setting the genetic code in all living organisms and are found in all three domains of life Bacteria (B), Archaea (A), and Eukarya (E). The essential process of protein synthesis requires twenty sets of synthetases and their corresponding tRNAs for the correct transmission of the genetic information. The aaRSs are responsible for loading the twenty different amino acids (aa) onto their cognate tRNA (tRNA containing the appropriate anticodon). The formation of aminoacyl-tRNA (aa-tRNA) occurs via direct acylation or an indirect mechanism in which the amino acid or amino acid precursor in the misacylated tRNA is modified in a second step. These indirect pathways suggest interesting evolutionary links between amino acid biosynthesis and protein synthesis [8, 9].

Each aaRS is a multidomain protein consisting of (at least) a catalytic domain and an anticodon binding domain. In all known cases, the synthetases can be divided into two types based on homology of their catalytic domains: class I or class II. Class I aaRSs possess the basic Rossmann fold, while class II aaRSs exhibit a fold that is unique to them and biotin synthetase holoenzyme. Additionally, some of the aaRSs, for example the bacterial leucyl-tRNA synthetase, have an “insert domain” within their catalytic domain (see Figure 1). The tRNA is charged in the catalytic domain and recognition of it takes place through interactions with the anticodon loop, acceptor stem, and D-arm of the tRNA (see Figure 1). We will examine the evolution of the structure and sequences of the aaRSs.

1.3 Assembling the Tools for Network Analysis

1.3.1 Requirements

In order to carry out all of the network analyses and visualizations presented in this tutorial, several different applications need to be installed on your local

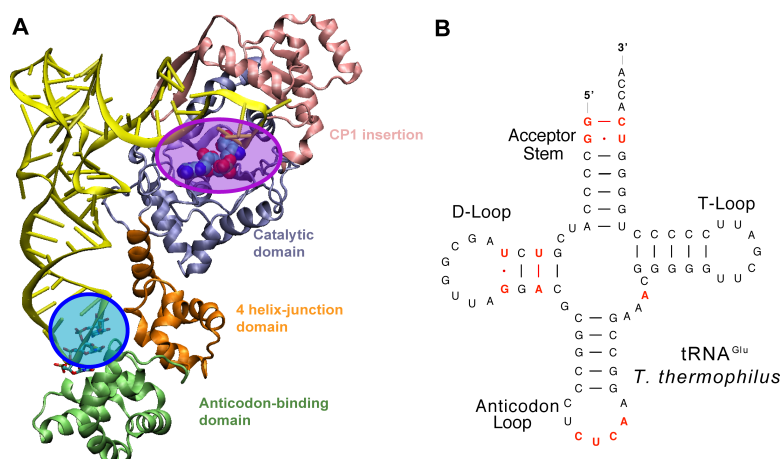


Figure 1: aaRS:tRNA complex **A**. A snapshot of GluRS:tRNA:Glu-AMP complex (from *T. thermophilus*; PDB code 1n78) in the active form. The tRNA (shown in yellow) is docked to GluRS (shown as cartoon), and the analog of Glu-AMP substrate is shown in space-filling representation. The GluRS can be divided into four parts: the anticodon-binding domain (green), the four helix-junction domain (orange), the CP1 insertion (purple), and the catalytic domain (blue). The catalytic active site is highlighted within the catalytic domain (pink oval); The three anticodon bases are also highlighted (blue oval). Note that specific contacts between the tRNA and GluRS allow for strategic positioning of the tRNA relative to the enzyme. **B**. The secondary structure of *T. thermophilus* tRNA^{Glu}. The bases that are essential for tRNA recognition by GluRS are shown in red.

computer and configured. Here are the necessary programs and their corresponding download locations:

- VMD 1.9.1 or later: <http://www.ks.uiuc.edu/Research/vmd/>
- Carma 0.8 or later: <http://utopia.duth.gr/~glykos/Carma.html>
- CatDCD 4.0 pre-compiled binary (not the plugin included with VMD): <http://www.ks.uiuc.edu/Development/MDTools/catdcd/>
- gncommunities and subopt are provided with the tutorial files, but you can also download them here: <http://www.scs.illinois.edu/schulten/software/networkTools/>

Install the above programs except for gncommunities and subopt which are provided with the tutorial files and discussed below. Each will have installation instructions provided by the developers.

This tutorial also requires a set of data files that serve as input to the network analysis and visualization software. These files as well as the pdf of this tutorial are available here: <http://www.scs.illinois.edu/schulten/tutorials/network/>

1.3.2 PATH Setup for Carma and CatDCD

Carma calculates covariance and correlation between pairs of atoms across an MD trajectory. It is used to generate the weights for the dynamical networks. After downloading Carma, it needs to be placed in your computer's PATH environment variable so that the program is found when it is called. CatDCD is used to create stripped-down MD trajectories for the Carma calculations. Since it is also called from within VMD, it needs to be placed in the PATH as well.

The process of adding programs to a computer's PATH is different on different operating systems. Follow the directions below for your specific operating system.

Linux or Mac OS X: First, you need to locate the directories containing the Carma and CatDCD binaries. In a terminal window, navigate to the program's location and type `pwd` to retrieve the full path. These two paths will be added to the end of the PATH variable.

Check which shell you are using by opening a terminal and typing:

```
echo $SHELL
```

If you are using bash, you can add a directory to your path by adding the following command to the `.bashrc` (Linux) or `.profile` (Mac OS X) file in your home directory:

```
export PATH=$PATH:/path_to_your_directory
```

For example, if the Carma binary is sitting in `/home/username/carma/bin/linux`, you should type:

```
export PATH=$PATH:/home/username/carma/bin/linux
```

Do this for both Carma and CatDCD.

If you are using csh, you can add a directory to your path by adding the following command to the `.bashrc` (Linux) or `.profile` (Mac OS X) file in your home directory:

```
set PATH = ($PATH /path_to_your_directory)
```

For example, if the Carma binary is sitting in `/home/username/carma/bin/linux`, you should type:

```
set PATH = ($PATH /home/username/carma/bin/linux)
```

Do this for both Carma and CatDCD.

Microsoft Windows 7:

On Windows, you must find and access a specific settings dialog to add a directory to your Path.

1. From the Start Menu select the Control Pad.
2. Select “System and Security”.
3. Select “System”.
4. To the left of the window under “Control Panel Home” select “Advanced System Settings”, and a dialog box titled “System Properties” should pop up with the “Advanced” tab open.
5. Click the “Environment Variables...” button at the bottom of this screen, and another dialog should pop up.
6. Under “System variables”, find and select the “Path” variable.
7. Click the Edit... button below the selector, and an “Edit System Variable” dialog will pop up with settings for the “Path” variable.
8. The Path “Variable value” is a set of directories separated by semicolons. Add the directories where you installed Carma and CatDCD.

Now your Path should be set up, and both Carma and CatDCD should be callable from the command line.

1.3.3 Move gncommunities and subopt into the common Directory

Underneath TUTORIAL_DIR/bin/ there are directories containing binary executables for gncommunities and subopt (gncommunities.exe and subopt.exe on MS Windows). Copy the binaries for your operating system into the TUTORIAL_DIR/common directory. For example, on Linux you could navigate to TUTORIAL_DIR/bin/Linux-x86-64 and run:

```
cp gncommunities subopt ../../common
```

or simply move the files by using your system’s file browser. The remainder of the tutorial assumes that you have working executables in the common directory.

1.4 Network File Preparation

To create a network model for our protein-RNA system, we first need to specify how the nodes and edges will be created. Typically, a node represents some set of atoms, e.g. an amino acid within a protein. You could have a node for every single atom in the system, but here we are looking for a coarse grained representation that makes the structures easier to think about. We will assign

one node to each amino acid in GluRS and two nodes to each nucleotide in the tRNA (one for the base and one for the sugar and phosphate). Amino acid nodes will be centered C_α atoms and the two nucleotide nodes will be located at nitrogens (N1 or N9) participating in the N-glycosidic bond and at the phosphorous atom. Since the adenylate in the active site is composed of AMP and glutamate moieties, we will use three nodes to represent the molecule.

The next step is to define edges between pairs of nodes. One way is to connect nodes that are within a certain distance of one another. Here we incorporate dynamics into our definition and draw edges between nodes whose residues are within a cutoff distance (4.5 Å) for at least 75% of an MD trajectory. Crystal contacts that disappear are ignored while contacts that form but are not present in the original structure are included.

The edge distances d_{ij} are derived from pairwise correlations (C_{ij}) which define the probability of information transfer across a given edge: $d_{ij} = -\log(|C_{ij}|)$ where $C_{ij} = \frac{\langle \Delta \vec{r}_i(t) \cdot \Delta \vec{r}_j(t) \rangle}{(\langle \Delta \vec{r}_i(t)^2 \rangle \langle \Delta \vec{r}_j(t)^2 \rangle)^{1/2}}$, $\Delta \vec{r}_i(t) = \vec{r}_i(t) - \langle \vec{r}_i(t) \rangle$, and $\vec{r}_i(t)$ is the position of the atom corresponding to the i^{th} node. Correlations will be calculated from an MD trajectory by the program Carma [10]. Principal component analysis can be carried out on this correlation matrix, but here we use the correlation data to weight edges in the dynamical network. A residue-residue correlation map for GluRS·tRNA^{Glu} is shown in Figure 2.

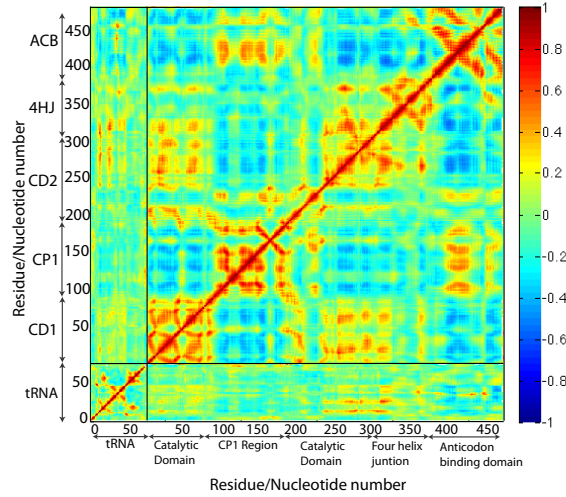


Figure 2: Cross correlation map for GluRS·Glu-tRNA^{Glu} based on the last 16 ns of a 20-ns simulation. From [11].

The configuration file (in TUTORIAL_DIR/1.network_preparation/network.config) for the dynamical network looks like this:

```
>Psf
../1.network_preparation/gluRStRNA.psf
```



```

>Dcds
../1.network_preparation/gluRStRNA.dcd

>SystemSelection
(chain P N X) and (not hydrogen)

>NodeSelection
(name CA P) or (resname CYT URA and name N1) or (resname ADE GUA GOM
and name N9)

>Clusters
N9 ADE name N9 C8 N7 C5 C6 N6 N1 C2 N3 C4
P ADE name P 01P 02P 05' C5' C4' 04' C1' C3' C2' 02' 03'
N1 CYT name N1 C2 02 N3 C4 N4 C5 C6
P CYT name P 01P 02P 05' C5' C4' 04' C1' C3' C2' 02' 03'
N9 GUA name N9 C8 N7 C5 C6 06 N1 C2 N2 N3 C4
P GUA name P 01P 02P 05' C5' C4' 04' C1' C3' C2' 02' 03'
N1 URA name N1 C6 C5 C4 04 N3 C2 02
P URA name P 01P 02P 05' C5' C4' 04' C1' C3' C2' 02' 03'
P GOM name P 01P 02P 03P 05' C5' C4' 04' C1' C3' C2' 02' 03'
CA GOM name N CA CB CG CD OE1 OE2 C O

>Restrictions
notSameResidue
notNeighboringCAlpha
notNeighboringPhosphate

```

The `Psf`, `Dcd`, `SystemSelection`, and `NodeSelection` fields are absolutely necessary. `Psf` and `Dcd` define the `psf` and `dcd` files used to create the network. `SystemSelection` and `NodeSelection` are VMD atomselection strings specifying two different sets of atoms. `SystemSelection` is used to determine the contact map. If atoms corresponding to a node (n_1) are within the contact distance of atoms from another node (n_2) for 75% of the trajectory, then the pairwise (n_1, n_2) correlation is kept, and an edge will be created between n_1 and n_2 in the network. `NodeSelection` is used to select only those atoms that will represent nodes in the network. A `dcd` file with these atoms is created and given to Carma so the resulting correlation values (and ultimately the network distances) come from the `NodeSelection` atoms. In `NetworkView`, the nodes will be placed at these atoms in the loaded structure.

The `Clusters` block is used to associate a set of atoms with a node. If no clusters are defined, a node corresponds to the atoms within a given residue. The form of cluster entries is “atomname resname selectionstring” where selectionstring identifies the atoms represented by the node.

`Restrictions` is a set of constraints that specify other edges that should be removed from the final network. “notSameResidue” disallows edges between

nodes in the same residue, “notNeighboringCAlpha” disallows edges between C-alpha atoms that have adjacent residue numbers, “notNeighboringPhosphate” disallows edges between P atoms that have adjacent residue numbers, and “notNeighboringResidue” disallows edges between nodes from adjacent residues.

1. Open VMD, and using the console, enter `TUTORIAL_DIR/1.network_preparation`.
2. Copy the network configuration file to your working directory: `cp network.config ../common` and then move to the working directory.
3. To create a dynamic network from the MD trajectory, type the following command at the VMD console: `networkSetup network.config`. This calculation will take a few minutes. During this process catdcd and Carma are called to process the trajectory and output pairwise correlations for node atoms.

The final output will be several ASCII files containing an atomselection string and matrices. The atomselection string is used by NetworkView to map network nodes onto the node atoms specified by `NodeSelection` in the network-Setup configuration file. Now you are ready to view the network in VMD.

2 Dynamical Network Representations

2.1 Load a network into NetworkView

Note that files needed for this tutorial are included in `TUTORIAL_DIR` under directories numbered by section. If you have trouble generating any output files or just want to skip certain steps, the final files are provided in these directories. Just copy the relevant files into your working directory (`TUTORIAL_DIR/common`).

1. If NetworkView is not running, start it from within VMD by choosing the **Extensions** menu and then selecting **Analysis** → **NetworkView**. The NetworkView program window will appear on your screen.
2. Choose the **File** menu and select **Load Network...** The Load Network window will appear.
3. Select the file from your local tutorial directory at `TUTORIAL_DIR/common/contact.dat`. Click the **Open** button to have NetworkView load the network into the currently active structure.

At this point, the network should appear on top of the structures in VMD. If you change the molecule representation from “Lines” to “Tube”, then the view should be similar to Figure 3.



Figure 3: Unweighted network for GluRS-tRNA^{Glu}.

2.2 Deactivate and activate subnetworks

Turn tRNA off and then back on.

1. Make sure the **Atom Selection** radio button is chosen under the section titled “Node Selection”.
2. Type “nucleic” in the text box to the right of **Atom Selection** so that only nodes within the tRNA are considered.
3. Choose **Deactivate** under the “Action” section.
4. Click the **Apply** button followed by the **Draw** button.

The subnetwork corresponding to tRNA^{Glu} should have disappeared leaving only the network for GluRS.

2.3 Color the network

1. Activate the entire network so all edges and nodes are shown.
2. Choose the **Color ID** radio button under “Action”, and then picking “red” from the menu to the right.
3. Make sure **Atom Selection** with text “all” is chosen under “Node Selection” before hitting the **Apply** button at the bottom.

4. Finally, hit the **Draw** button to update the network depiction in the OpenGL window.

Some changes to the network representation take some time. The **Apply** and **Draw** buttons are separate so that the OpenGL window is not updated every time a small change is made. Go ahead and color the protein blue and the tRNA green by repeating the same procedure with “protein” and “nucleic” as the respective atomselection strings. You should notice that the edges bridging the interface between the GluRS and tRNA^{Glu} remain red.

2.4 Weight the network using correlation data



Figure 4: Network for GluRS-tRNA^{Glu} shown with edge widths corresponding to their weights ($-\log(|C_{ij}|)$).

1. First set color of the entire network back to “blue”.
2. Under the “Display Parameters” section choose the **weight** option for the **Edge Size**.
3. Click **Draw** to update the display.

With the weight information displayed, the network should appear similar to the one shown in Figure 4.

3 Communities

3.1 Run gncommunities to generate network communities

The dynamical network can be used as a basis for further analysis. First we will look at the community substructure of the network which we obtain by applying the Girvan-Newman algorithm [6]. Communities are subnetworks that partition the original network. Nodes in a community have more and stronger connections within that community than to nodes in other communities. With our correlation-based weights, communities correspond to sets of residues that move in concert with each other. Each node is necessarily part of a community even if it just a community of one, but there are edges that lie between communities connecting the nodes of one community to those of another.

1. To calculate the community partitioning of our system, go to the terminal and enter the directory `TUTORIAL_DIR/common`.
2. The `gncommunities` program should be in `TUTORIAL_DIR/common`. Run `gncommunities` on the dynamical network: `./gncommunities contact.dat communities.out`
3. You can see a description of the `gncommunities` input parameters by running the program with no parameters: `./gncommunities`

Four files are generated: `betweenness.dat`, `communities.out`, `Community.tcl`, and `output.log`. `communities.out` is needed by `NetworkView` to manipulate and display communities within `VMD`.

3.2 Load community data files

1. Make sure that `VMD` is open, and you have `gluRStRNA.psf` and `gluRStRNA.pdb` loaded as well as the network data (`contact.dat`).
2. From the `NetworkView` window choose the `File` menu and select `Load Community Data....` The `Load Communities` window will appear.
3. Select the file from your local tutorial directory at `TUTORIAL_DIR/common/communities.out`. Click the `Open` button to have `NetworkView` load the community information into the currently active structure.

3.3 Activate and color by community

Now information about the community structure of `GluRS-tRNAGlu` is available for manipulation within `NetworkView`. Individual communities can be selected for activation and coloring. Selections can also be made for groups of communities. Select a specific community and color it separately.

1. Select `Community` under “Node Selection” and highlight community number 8.



Figure 5: Network for GluRS·tRNA^{Glu} colored by community.

2. To color this community select **Color ID**, set the color to “lime”, click **Apply** and then **Draw**.
3. To view only this community, deactivate the entire network then activate community 8 by choosing **Activate** and hitting **Apply** and then **Draw**.

Coloring each community a different color can be time consuming, especially when there are many communities present. To color all communities at once, you can use the **Color Communities** option.

1. First activate all communities by hitting **All** to the right of the **Community** option.
2. Activate this selection. Notice how the edges between communities were not activated. When using community selections, actions are performed for each community separately so edges between communities are ignored.
3. To color all communities select **Color Communities**, click **Apply** and then **Draw**.

Your display should look similar to Figure 5 with each community having a different color. However, the communities in the figure have been manually colored so that similar colors are not too close to each other. Color contrast can be important when making figures, but in everyday use **Color Communities** is handy and usually sufficient.

3.4 Examine critical nodes and edges between communities

Critical nodes connect communities so they lie in the interface between pairs of communities. Here we color the critical nodes and edges red.

1. Choose the Critical Node radio button under “Node Selection”.
2. Select Color ID “red” under “Action”.
3. Finally, hit Apply and then Draw.

These nodes and edges are defined based on a network metric called betweenness. The betweenness of an edge is the number of pairwise optimal paths that cross that edge. Betweenness is called a “centrality” measure as it shows how important an edge is to the entire network.

4 Optimal and Suboptimal Paths

4.1 Run subopt to obtain suboptimal paths

Suboptimal paths are generated by the program subopt from the initial dynamical network matrix. Two nodes are chosen: a source and a sink. Suboptimal paths are defined as paths that are slightly longer than the optimal path so we need to specify how much longer they can be. A given suboptimal path will not visit any node more than once.

The subopt program takes the source and sink nodes as parameters so you will need to obtain their ID numbers. This can be done through the TkConsole in VMD using one of the command line procedures. Here, you will look at the suboptimal paths between the base of U35, the middle nucleotide of the anticodon, and the sugar of A76 which is charged by glutamate. It doesn’t actually matter which node is the source and which is the sink because the suboptimal path determination is symmetric with respect to the end nodes.

1. To find the base node for U35, go to the TkConsole and type:

```
::NetworkView::getNodesFromSelection "chain N and resid 535 and name N1"
```

 This returns the node ID for the base (should be 69). Above, in the network configuration file, the N1 atom on uracil was chosen as the location for the placement of the base node.
2. The sugar and phosphate of nucleotides were combined into a single node located at the phosphorous atom. To retrieve the A76 sugar node, type:

```
::NetworkView::getNodesFromSelection "chain N and resid 576 and name P"
```

 This should return node ID 148.

3. To calculate some suboptimal paths for our system, go to the terminal and make sure you are in the directory TUTORIAL_DIR/common.
4. The subopt program should be in TUTORIAL_DIR/common. Run subopt on the dynamical network: `./subopt contact.dat u35-a76.out 20 69 148`
This calculates suboptimal paths from a node in the anticodon to A76 in the active site.
5. You can see a description of the subopt input parameters by running the program with no parameters: `./subopt`

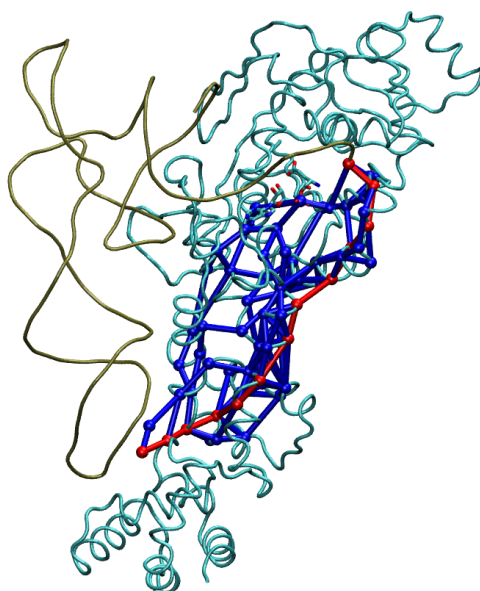


Figure 6: Suboptimal paths are shown between the base of U35 and the sugar of A76 on the tRNA. All paths are colored blue except for the optimal path which is red.

4.2 Load suboptimal path files

Once the suboptimal paths have been calculated, you can load this new information into NetworkView and check out the bundle of paths connecting the two residues.

1. Choose the File menu and select Load Suboptimal Path Data....
2. Select the file from your local tutorial directory at TUTORIAL_DIR/common/u35-a76.out. Click the Open button to have NetworkView load the suboptimal path information into the currently active network.

You will now see information in the “Node Selection” section corresponding to the loaded suboptimal paths. Multiple sets of suboptimal paths can be calculated and then loaded into `NetworkView` simultaneously. The menu next to the **Suboptimal Path** radio button sets the current suboptimal path set with a given source and sink. The pick list underneath contains all suboptimal paths in that set. The first path listed is actually the optimal path.

4.3 Activate and color the suboptimal paths

1. Set the entire network’s color back to blue.
2. Deactivate the whole network.
3. To activate the suboptimal paths, choose **Suboptimal Path** under “Node Selection”. Only one option will be available in the Suboptimal Path menu.
4. Click on the associated **All** button to the right to highlight all of the suboptimal paths.
5. Make sure that **Activate** is chosen under the “Action” settings.
6. Hit **Apply** and then **Draw**. These steps may take a few seconds to complete.
7. To highlight the optimal path, select “Suboptimal Path” 0, and change its color to red.

4.4 Load suboptimal path count into value

1. Select “Load into value” under the “Action” settings, make sure the associated menu selection is “suboptPathCount”, and hit **Apply**.
2. To visualize this new data, set “Edge Size” to “value” and hit **Draw** to redraw the network.
3. The nodes might obscure the current view of the edges. Deactivate the nodes to get them out of the way.

Now the suboptimal path view shows each edge weighted by the number of suboptimal paths that cross that edge. The optimal path should contain relatively thick edges compared to the rest of the edges shown in the suboptimal paths.

5 Application Programming Interface (API)

5.1 Beyond the Graphical User Interface (GUI)

So far we have explored how to interact with networks primarily through the `NetworkView` window, which is accessible from the VMD Extensions menu. As

seen in the previous section, there are other, more complex procedures only available through the command line. These procedures are particularly useful for obtaining detailed information about loaded networks. Once this information has been extracted, it can be further analyzed using the Tcl programming language.

We will now look at three `NetworkView` procedures: `getInterfaceEdges`, `getEdgeInfo`, and `getEdgesByMetric`. This work will be done through the `VMD TkConsole`.

5.2 `getInterfaceEdges`

The first procedure provides a way to retrieve edges that connect one subnetwork to another. This allows you to focus on regions of the network such as contacts between members of a macromolecular complex or interactions between an active site and a bound ligand. Each of the two subnetworks is defined by its set of nodes. In this case, the relevant node indices are most easily retrieved through calls to `getNodeFromSelection`, an API command we used in the previous section.

1. Type commands into the `TkConsole` to retrieve the node indices for the `GluRS`:

```
set glursNodes [::NetworkView::getNodeFromSelection "chain P"]
```

and then for the `tRNA`:

```
set trnaNodes [::NetworkView::getNodeFromSelection "chain N"]
```
2. To get the set of edges at the `aaRS:tRNA` interface, enter the following in the `TkConsole`:

```
set interfaceEdges [::NetworkView::getInterfaceEdges $glursNodes $trnaNodes]
```

`getInterfaceEdges` returns a set of `nodeId` pairs that define the edges and stores them in the `interfaceEdges` variable.
3. Now that you have the edges stored in the `interfaceEdges` variable, you can change their color by passing them to the edge coloring procedure:

```
::NetworkView::colorEdges $interfaceEdges red
```
4. To update the display, either click the “Draw” button or use the “Draw” API command:

```
::NetworkView::drawNetwork
```

5.3 `getEdgeInfo`

Each edge has an associated set of information that can be queried from the command line. Two fields that we have seen already are “weight” and “value”. These numeric data can be retrieved for any given set of edges.

1. Get the weights for edges at the aaRS:tRNA binding interface:

```
set edgeWeights [::NetworkView::getEdgeInfo weight $interfaceEdges]
```
2. We can now find the mean edge weight across the binding interface.

```
vecmean $edgeWeights
```
3. Combining information from the weight list with the edge list, we can determine which interface edge has the smallest weight. First sort the list of weights to find the value of the minimum weight.

```
lsort $edgeWeights  
lsearch -regexp $edgeWeights {0\.24}  
lindex $interfaceEdges 81
```
4. Color this edge green so that you can see where it is in the context of the system.

```
::NetworkView::colorEdges [list 69,593] green  
::NetworkView::drawNetwork
```

5.4 getEdgesByMetric

It can be useful to filter the network information by weight or value. The strongest edges can be selected using the procedure `getEdgesByMetric`.

1. To select just the strongest edges, those with the lowest weights:

```
::NetworkView::deactivateEdgeSelection "all"  
::NetworkView::drawNetwork  
set strongEdges [::NetworkView::getEdgesByMetric weight 0 0.2]  
::NetworkView::activateEdges $strongEdges  
::NetworkView::drawNetwork
```

How are the strong edges distributed throughout the network? To what sorts of structures do they belong?

Aside from these three command-line functions, all of the other functionality, such as network loading and drawing, is available through the API.

6 Acknowledgements

Development of this tutorial was supported by the National Institutes of Health (P41-RR005969 - Resource for Macromolecular Modeling and Bioinformatics) and the National Science Foundation (MCB-0844670).

References

- [1] A. Sethi, J. Eargle, A.A. Black, and Z. Luthey-Schulten. Dynamical networks in tRNA: protein complexes. *Proc. Natl. Acad. Sci. USA*, 106(16):6620–6625, 2009.
- [2] Alexis Black Pyrkosz, John Eargle, Anurag Sethi, and Zaida Luthey-Schulten. Exit strategies for charged tRNA from GluRS. *J. Mol. Biol.*, 397:1350–1371, Apr 2010.
- [3] A. del Sol, H. Fujihashi, D. Amoros, and R. Nussinov. Residues crucial for maintaining short paths in network communication mediate signaling in proteins. *Mol Syst Biol*, 2:1–12, Jan 2006.
- [4] Amit Ghosh and Saraswathi Vishveshwara. A study of communication pathways in methionyl- trna synthetase by molecular dynamics simulations and structure network analysis. *Proc. Natl. Acad. Sci. USA*, 104(40):15711–6, 2007.
- [5] Chakra Chennubhotla and Ivet Bahar. Markov propagation of allosteric effects in biomolecular systems: application to groel-groes. *Mol. Syst. Biol.*, 2:36, Jan 2006.
- [6] M Girvan and M Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99:7821–7826, 2002.
- [7] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–8, Jun 2005.
- [8] Patrick O’Donoghue, Anurag Sethi, Carl R Woese, and Zaida A Luthey-Schulten. The evolutionary history of Cys-tRNA^{Cys} formation. *Proc. Natl. Acad. Sci. USA*, 102(52):19003–19008, Dec 2005.
- [9] Anselm Sauerwald, Wenhong Zhu, Tiffany A Major, Herve Roy, Sotiria Palioura, Dieter Jahn, William B Whitman, John R. Yates 3rd, Michael Ibba, and Dieter Söll. RNA-dependent cysteine biosynthesis in archaea. *Science*, 307:1969–1972, Mar 2005.
- [10] Nicholas M Glykos. Software news and updates. CARMA: a molecular dynamics analysis program. *J. Comp. Chem.*, 27(14):1765–1768, Nov 2006.
- [11] R.W. Alexander, J. Eargle, and Z. Luthey-Schulten. Experimental and computational determination of tRNA dynamics. *FEBS Lett.*, 584(2):376–386, 2010.