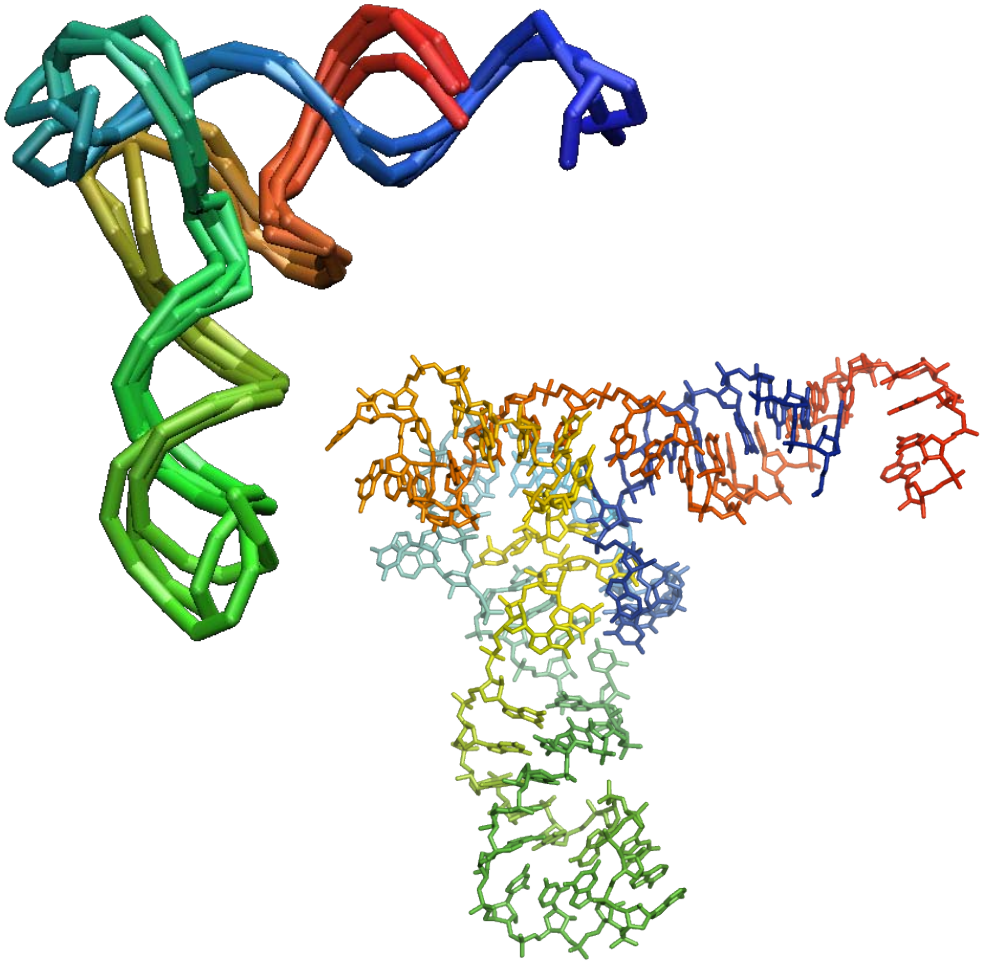
NAST/C2A



Tutorials

Release 0.5

June 16, 2009

Website: Simtk.org/home/nast

Copyright and Permission Notice

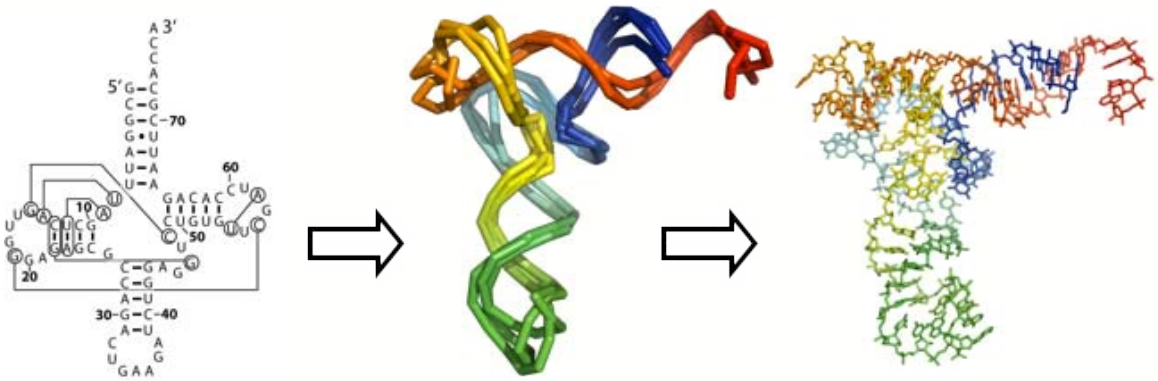
Acknowledgments

NAST/C2A and all related activities are funded by the Simbios National Center for Biomedical Computing through the National Institutes of Health Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers can be found at http://nihroadmap.nih.gov/bioinformatics.

Table of Contents

1 Overview

NAST/C2A is a set of Python tools that enables you to generate full-atomic 3D RNA structures from secondary structure information in less than a day. NAST generates coarse-grained 3D structures from secondary structure information, and C2A adds the full-atomic details to these coarse-grained models.



1.1 Nucleic Acid Simulation Toolkit (NAST)

NAST is a knowledge-based coarse-grained tool for modeling RNA structures. It produces a diverse set of plausible 3D structures that satisfy user-provided constraints based on:

1. Primary sequence
2. Known or predicted secondary structure
3. Known or predicted tertiary contacts (optional)

Additionally, NAST can use residue-resolution experimental data (e.g., hydroxyl radical footprinting) to filter the generated decoy structures. By filtering the coarse-grained 3D

structures it produces based on agreement to available experimental data, a model of the molecule which satisfies all the known residue-resolution data is produced.

NAST is written in Python and incorporates Python-OpenMM, a Python version of OpenMM, a library that allows molecular dynamics simulations to be accelerated on graphics processing units (GPUs).

Please cite the following article in any published work which utilizes NAST:

Jonikas MA, Radmer RJ, Laederach A, Das R, Pearlman S, Herschlag D, Altman RB. Coarse-grained modeling of large RNA molecules with knowledge-based potentials and structural filters. RNA. 2009 Feb;15(2):189-99.

1.2 Coarse to Atomic (C2A)

C2A uses a knowledge-based approach to instantiate full atomic detail into coarse grain templates of 3D RNA structures. C2A uses geometries observed in known RNA 3D crystal structures to find plausible full atomic matches to fragments in a coarse-grained template structure. Currently, only coarse grain models that use a one-point-per-residues (the C3' atom) representation can be used as input. Models constructed using NAST can be run through C2A to generate a full-atomic model. C2A is written in Python.

Please cite the following in any published work which utilizes C2A:

Jonikas M.A., Radmer R.J., Altman R.B. Knowledge-Based Instantiation of Full Atomic Detail into Coarse Grain RNA 3D Structural Models. Submitted to Bioinformatics.

http://simtk.org/home/nast

2 Prerequisites

NAST/C2A: You can download NAST/C2A from http://simtk.org/home/nast`. Click on "Downloads" and follow the directions in README.pdf to install and test NAST/C2A.

Example files: On Windows, you will need to download a separate package for the examples, also available from http://simtk.org/home/nast. Example files for NAST/C2A are included in the Mac and source code installation packages.

VMD (or another software for viewing PDB-format structures): Download VMD from http://www.ks.uiuc.edu/Research/vmd. Click on "Download VMD" and select the installation for your platform. We recommend you get Version 1.8.6 or higher. Follow the on-line instructions for installing.

3 The Basics of Running NAST

3.1 Objectives

These exercises are intended for you to:

- Learn the basics of how to run NAST
- Learn how to visualize the NAST results within VMD

3.2 Open a command prompt/terminal window

NAST is run from the command prompt/terminal. To launch a command prompt/terminal window, select:

(Windows) Start -> All Programs -> Accessories -> Command Prompt
(Mac OS) Macintosh HD -> Applications -> Utilities -> Terminal

3.3 Navigate to your examples folder

Within the command prompt/terminal window, navigate to the nast-0.5 examples folder The exact directory path to this folder will vary, depending on where you saved the files you downloaded.

To change to another directory, use the command:

```
cd <directory path>
```

On Windows, for example, you downloaded a separate file for the examples. If you put the examples folder on your Desktop, then to get to the NAST/C2A examples folder, in the command prompt window, you would type:

(Windows) `cd "c:\Documents and Settings\Your Username\Desktop\nast-0.5.examples\nast"`

where you substitute `Your Username` with the name of your Windows log-in name.

Note: Quotation marks are required in specifying directory paths within the Windows command prompt window if the directory path includes spaces.

On Mac OS and Linux, the example files came with the programs and are located in a subdirectory of the NAST (Mac OS) or nast-0.5 (Linux) folders. So, for example, on the Mac, if you installed NAST on your Desktop, you would type the following in the terminal window:

(Mac OS) `cd /Users/<user_name>/Desktop/NAST/nast-0.5/examples/nast`

3.4 Your first NAST run

Let's try out NAST and see what happens.

1. Go to the 6TNA_MD example folder by typing:

   (Windows) `cd 6TNA_MD`
   (Mac OS/Linux) `cd 6TNA_MD`

2. Now run the test example `runTest.py` in this directory by typing:

   (Windows) `\Python26\python runTest.py`
   (Mac OS/Linux) `python runTest.py`

   Two files will be generated and saved in this directory: *6TNA_nast.pdb* and *6TNA_nast.psf*.

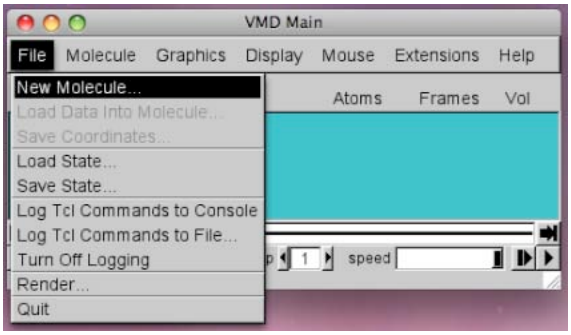
3. We can visualize the results within VMD:

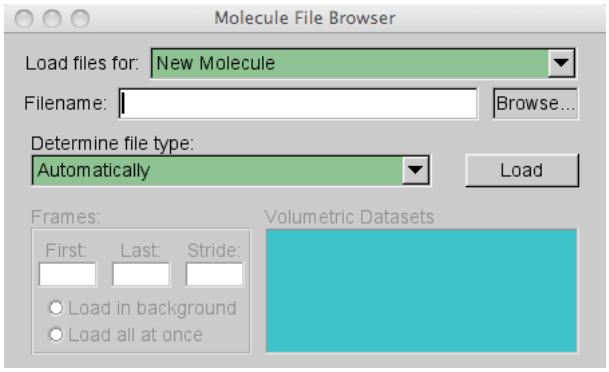
   a. Launch VMD. If you installed VMD in typical locations, you would select:

   (Windows) Start -> All Programs -> University of Illinois -> VMD -> VMD 1.8.6
   (Mac OS) Macintosh HD -> Applications -> VMD

   b. The "VMD Main" window will appear. Select:

   File -> New Molecule...





   c. To load the test trace into VMD, in the "Molecule File Browser" that appears, click on "Browse" and select the *6TNA_nast.pdb* created by NAST.

Note: In some Windows environments, the extensions (e.g., .pdb) will not appear. You can identify the PDB file by its icon (i.e.,  *) or by hovering over a file to see the file type.*

Click "Load" in the "Molecule File Browser."

d. A .psf file is needed to connect the residues together. To load this file in, return to the "Molecule File Browser" window. Set the fields as follows:

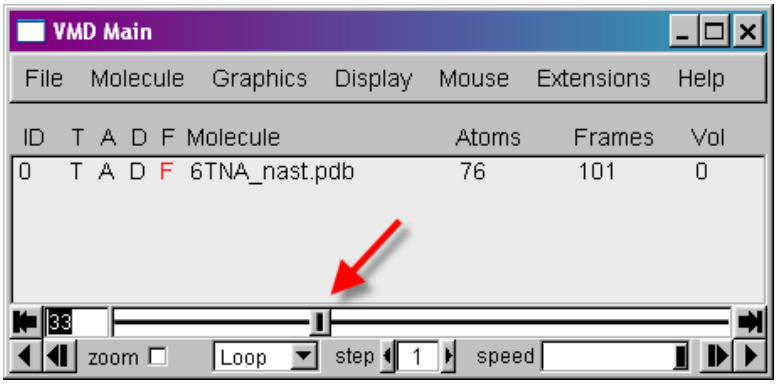
Load files for: 6TNA_nast.pdb
Filename: 6TNA_nast.psf

Note: In some Windows environments, the extensions (e.g., .psf) will not appear. You can identify the PSF file by its icon (i.e., 

) or by hovering over a file to see the file type.

Click "Load."

Scroll through the frames of the trajectory to see different conformations of one particular molecule using the slider bar in VMD (see figure below).



The structures should resemble those shown in the figure below. This particular run began with a coarse-grained representation of a crystal structure (see frame 0 of the trajectory). The first several frames of the simulation may look unusual, as the structure seeks to satisfy NAST's energy function. Although you are starting from a crystal structure, it may contain distances, angles or dihedrals that are not "RNA-like" based on the observations used to define the NAST energy function. In the process of satisfying the desired distribution of distances, angles, dihedrals and ideal helical geometry, the structure may pass through high energy states that do not look "RNA-like" before settling into an "RNA-like" conformation. If the structure does not settle quickly into a "RNA-like" conformation, it may be necessary to decrease the time-step of the simulation (more details on how to do this later).

3.5 Starting a NAST simulation from an unfolded state

In the previous example, the initial structure used for the NAST simulation was a crystal structure, so a 3D structure was already known. The primary use of NAST, though, is to generate 3D structures from an unfolded state.

In this exercise, you will use NAST to produce 3D structures from sequence data, secondary structure information, and tertiary contact information. We will also examine the details of the Python script used to accomplish this.

1. Copy the *runNast.py* Python script to *myRunNast.py* by typing the following into your command prompt/terminal window:

(Windows) `copy runNast.py myRunNast.py`

(Mac OS/Linux) `cp runNast.py myRunNast.py`

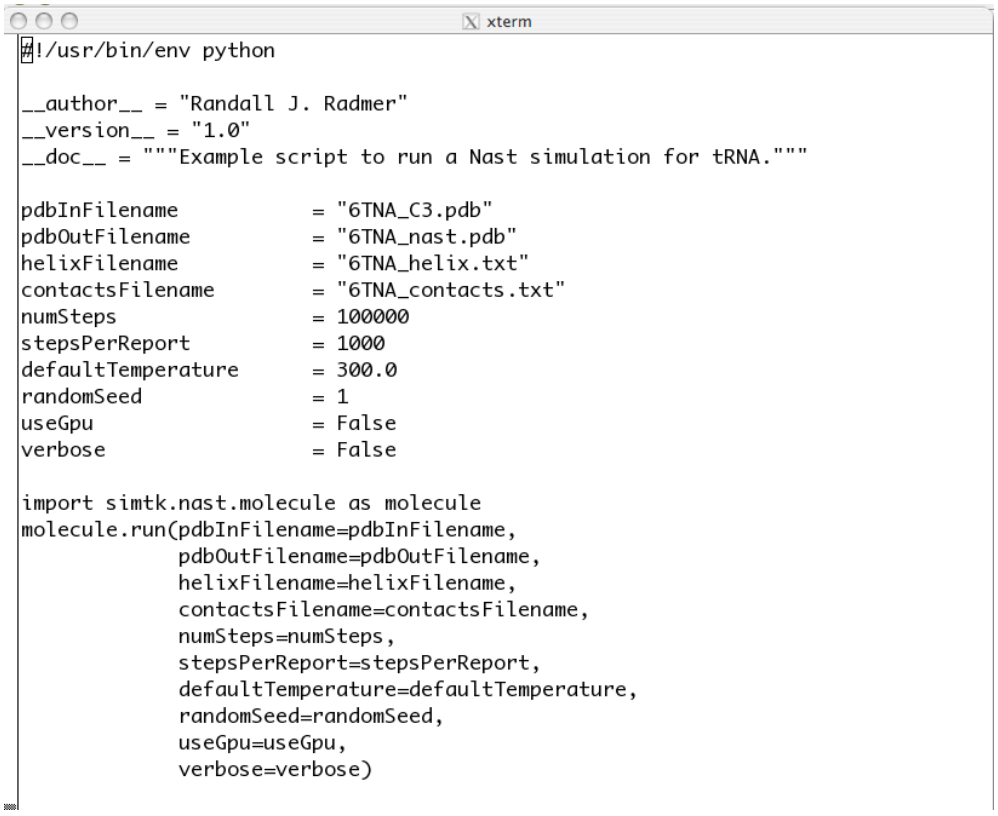
2. Open a text editor to edit *myRunNast.py*. DO NOT use Microsoft Word or other similar programs. They will insert formatting instructions that are not readable by NAST.

   On Windows, we recommend WordPad:

   (Windows) Start -> All Programs -> Accessories -> WordPad

   On Mac OS and Linux, there are many options, including emacs, vi, and TextEdit (Macintosh HD -> Applications -> TextEdit).

   You should see a file, like that shown below.



Parameter	**Details**
pdbInFilename	Specifies the input file name for the initial structure for the simulation. This can be a PDB file (must have extension .pdb) or a primary sequence – see details later in this example, as well as in Appendix A. Remember to put quotes around the file name.
pdbOutFilename	Specifies the name of the output trajectory file. Make sure to include the extension (.pdb). Remember to put quotes around the file name.
helixFilename	Specifies the name of the file that specifies the secondary structure. Remember to put quotes around the file name. Details about the file can be found in Appendix A.
contactsFilename	Specifies the name of the file that specifies the tertiary contacts. Remember to put quotes around the file name. Details about the file can be found in Appendix A.
numSteps	Specifies the total number of steps to run in the molecular dynamics simulation . This would be the maximum number of conformations you would obtain. Each step of the simulation represents 5 fs. Do **not** use commas or periods in specifying this number.
stepsPerReport	Specifies how often to save a frame (a conformation). In this example, with 1000 stepsPerReport and a numSteps of 100000, a total of 100 frames (conformations) would be saved. Do **not** use commas or periods in specifying this number.

Parameter	**Details**
defaultTemperature	Specifies the temperature for the molecular dynamics simulations in Kelvins. 300.0 K is the temperature at which NAST which characterized, so this is the value you will want to use unless you are purposefully trying to increase or decrease the energy of the system.
randomSeed	Specifies the random seed to be used by the molecular dynamics simulation program. You can give a specific value or the value of None, in which case the simulation will randomly choose a number. We recommend you use the value None (no quotes; case-sensitive).
useGpu	Specifies whether or not to use the GPU to accelerate the molecular dynamics. Valid values are `True` or `False`.
verbose	Specifies whether or not to print out the status updates as the simulation runs. Valid values are `True` or `False`.

3. Edit the script to:

    a. Start NAST from a sequence instead of a structure. Change the input file to *6TNA_C3.seq* as the input file. Primary sequence files cannot have the extension .pdb, which is reserved for structure inputs to NAST.

    b. Save the output to a new file: *T1.pdb*.

    c. Change the `randomSeed` value to `None` (capital N, followed by the lower case letters o, n, e).

```
xterm
#!/usr/bin/env python

__author__ = "Randall J. Radmer"
__version__ = "1.0"
__doc__ = """Example script to run a Nast simulation for tRNA."""

pdbInFilename           = "6TNA_C3.seq"
pdbOutFilename          = "T1.pdb"
helixFilename           = "6TNA_helix.txt"
contactsFilename        = "6TNA_contacts.txt"
numSteps                = 500000
stepsPerReport          = 1000
defaultTemperature      = 300.0
randomSeed              = None
useGpu                  = False
verbose                 = True

import simtk.nast.molecule as molecule
molecule.run(pdbInFilename=pdbInFilename,
             pdbOutFilename=pdbOutFilename,
             helixFilename=helixFilename,
             contactsFilename=contactsFilename,
             numSteps=numSteps,
```

4. Now run the edited script `myRunNast.py` by typing:

   (Windows) `\Python26\python myRunNast.py`
   (Mac OS/Linux) `python myRunNast.py`

   Two files will be generated and saved in this directory, based on the `pdbOutFilename` that you specified: *T1.pdb* and *T1.psf*.

5. Visualize trace *T1.pdb* in VMD (See test run exercise – Section 3.4). Remember to load in the *T1.psf* file to connect the residues.

   If you scroll through the frames of the trajectory, you will notice that in this example, you started from an unfolded circle. After a few frames, the structure took on a more RNA-like 3D structure (see figure below). Throughout the course of the simulation, the secondary structure and tertiary contacts are constrained. Every time the

simulation is run, the outcome will be different (provided the `randomSeed` variable is different) because there is a random component to the assignment of initial velocities.

3.6 Running a longer NAST simulation and creating initial files to be used with C2A

In this example, you will explore three other parameters in the Python script for running NAST (`numSteps`, `randomSeed`, and `verbose`) and learn to use VMD to generate the initial file needed by C2A.

1. Open a text editor to edit *myRunNast.py*. (see previous example on starting NAST from an unfolded state)

2. Make the following changes to *myRunNast.py*:

    a. Change the output file name to "`T4.pdb`"
    b. Change `numSteps` to `200000`
    c. Change `randomSeed` to `None`
    d. Optional: Change `verbose` to `True`

    In this example, we have increased the maximum number of timesteps for the molecular dynamics simulation (`numSteps`). This will increase the conformational diversity. Another way to achieve this is to repeat the run, but make sure you have set `randomSeed` to `None` so that there is randomness to your simulation. In this

example, we have done both. Note that if you set `randomSeed` to a particular value, the molecular dynamics simulation will be the same (as long as all the other variables are unchanged as well) since the velocities will be set based on the same seed value.

In this example, we also set `verbose` to `True` so that you can track the progress of the simulation.

If your simulation is taking too long to run, you can kill the job using **ctrl-z**. The frames that we generated thus far in the simulation will be saved.

3. Now run the edited script `myRunNast.py` by typing:

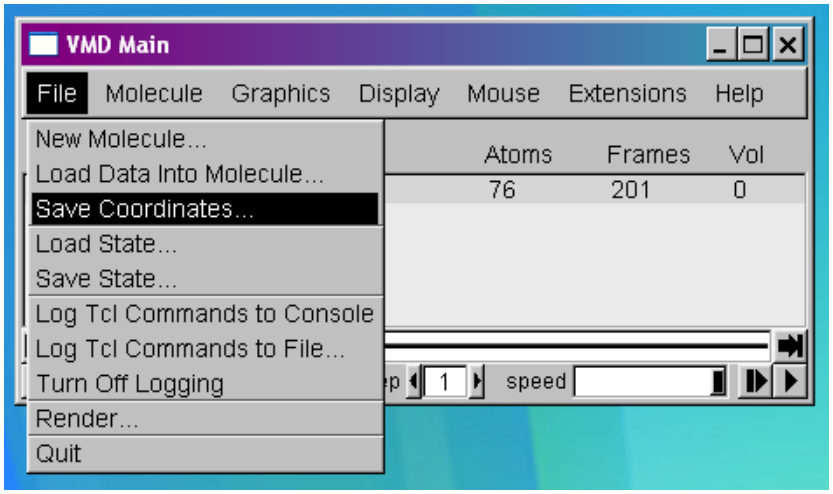
   (Windows) `\Python26\python myRunNast.py`
   (Mac OS/Linux) `python myRunNast.py`

   Two files will be generated and saved in this directory, based on the `pdbOutFilename` that you specified: *T4.pdb* and *T4.psf*.

4. Visualize trace *T4.pdb* in VMD (See test run exercise – Section 3.4). Remember to load in the *T4.psf* file to connect the residues.

5. From VMD, save one of the frames of *T4.pdb* as *T4-last.pdb*. Pick a frame that you will want to add full atomic detail to in a later exercise. It does not necessarily need to be the last frame of the simulation:

   a. Click on the trajectory of interest in the VMD Main Menu to select it.
   b. Select File -> Save Coordinates.



c. In the "Save Trajectory" window that appears, set:

Selected atoms: all

File type: pdb

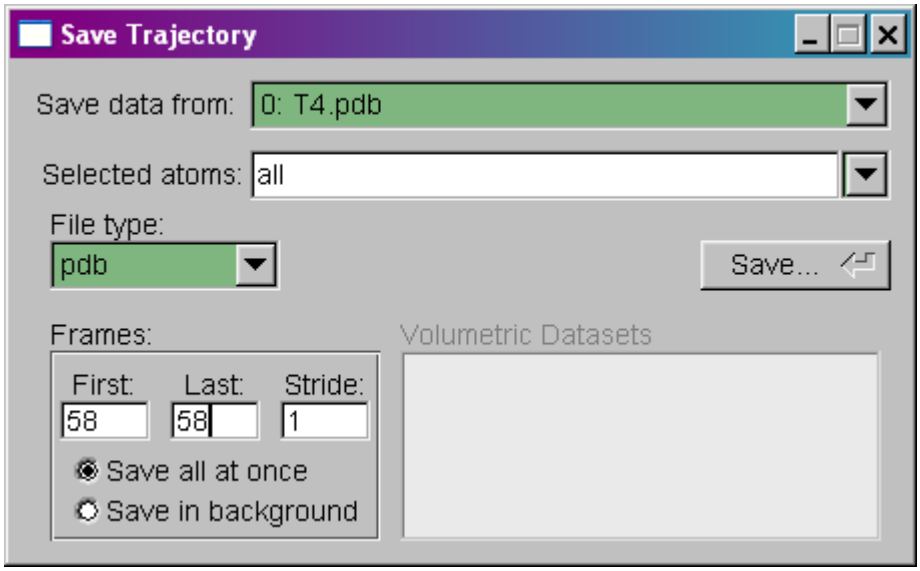
First: <frame number for the conformation to use with C2A>

Last: <same frame number as for First:>

Click "Save." When prompted for the file name to save to, enter:

`T4-last.pdb`

Be sure to specify the `.pdb` extension.



4 The Basics of Running C2A

4.1 Objectives

These exercises are intended for you to:

- Learn how to use C2A to search a reference molecule for fragments that match your coarse-grained molecule
- Learn how to use C2A to assemble fragment matches into a full atomic model

4.2 Open a command prompt/terminal window

C2A is run from the command prompt/terminal. To launch a command prompt/terminal window, select:

(Windows) Start -> All Programs -> Accessories -> Command Prompt
(Mac OS) Macintosh HD -> Applications -> Utilities -> Terminal

4.3 Navigate to your examples folder

Within the command prompt/terminal window, navigate to the nast-0.5 examples folder The exact directory path to this folder will vary, depending on where you saved the files you downloaded.

To change to another directory, use the command:

```
cd <directory path>
```

On Windows, for example, you downloaded a separate file for the examples. If you put the examples folder on your Desktop, then to get to the NAST/C2A examples folder, in the command prompt window, you would type:

(Windows) `cd "c:\Documents and Settings\Your Username\Desktop\nast-0.5.examples\nast"`

where you substitute `Your Username` with the name of your Windows log-in name.

Note: Quotation marks are required in specifying directory paths within the Windows command prompt window if the directory path includes spaces.

On Mac OS and Linux, the example files came with the programs and are located in a subdirectory of the NAST (Mac OS) or nast-0.5 (Linux) folders. So, for example, on the Mac, if you installed NAST on your Desktop, you would type the following in the terminal window:

(Mac OS) `cd /Users/<user_name>/Desktop/NAST/nast-0.5/examples/nast`

4.4 Finding fragment matches

The first step in adding full atomic detail to a coarse-grained model is to search a reference molecule for matching fragments and create a working library to be used by C2A. As with NAST, the process is controlled via a Python script.

1. First, change to the directory 6TNA_c2a.

   If you are continuing from the NAST examples, you would type the following into your command prompt/terminal window:

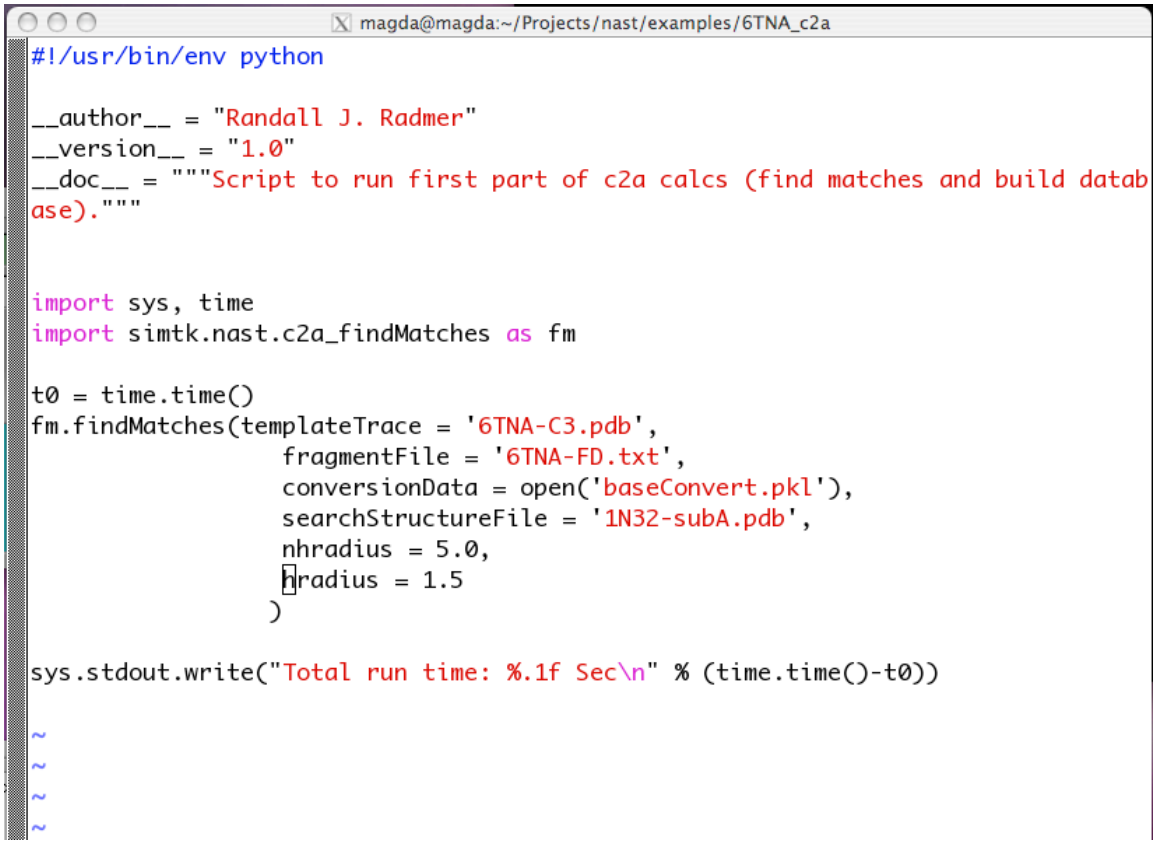
   (Windows) `cd ..\6TNA_c2a`
   (Mac OS/Linux) `cd ../6TNA_c2a`

If you are starting from the *examples* directory, type the following:

(Windows) `cd 6TNA_c2a`
(Mac OS/Linux) `cd 6TNA_c2a`

2. Open a text editor to examine the file *ex1.py*. DO NOT use Microsoft Word or other similar programs when editing these files, since they insert formatting instructions that are not readable by C2A.

   You should see a file like that shown below. This file is the script that searches a reference molecule for fragment matches and creates a working library that will be used later to C2A. The table below describes each of the 6 parameters for the script in more detail.



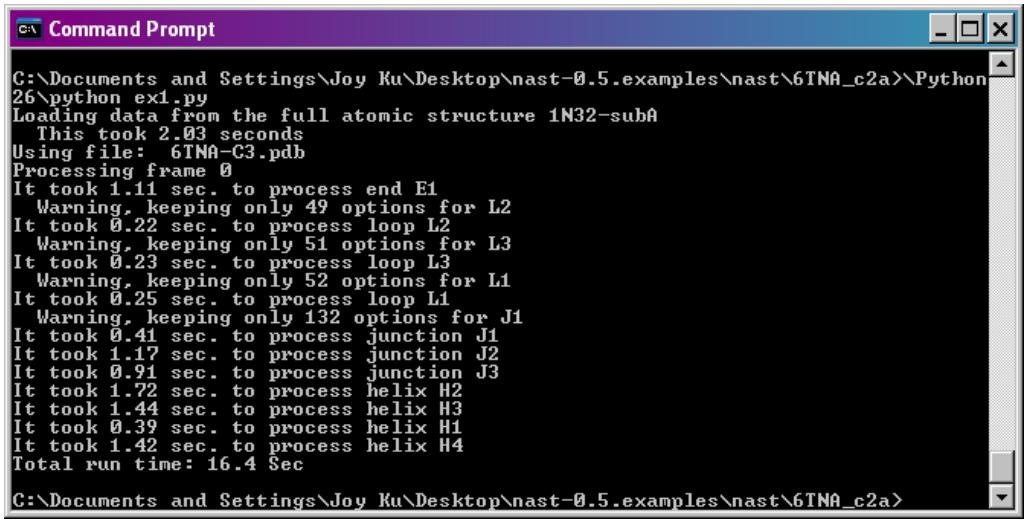
Parameter	**Details**
templateTrace	Specifies the name of the PDB file containing the coarse-grained model. Currently, only coarse-grained models with a representation of one-point-per-residue using the C3' atom are supported. Remember to put the file name in *single* quotes.
fragmentFile	Specifies the name of the text file that defines the fragments of the coarse-grained model that need to be matched. See Appendix B for more details. Remember to put the file name in *single* quotes. NOTE: This file name **must** be of the format *<molecule-name>-FD.txt*. NOTE: There **must** be a matching file which contains the primary sequence and is named *<molecule-name>-primary.txt* or *<molecule-name>.seq*.
conversionData	Specifies the name of the file that contains information for converting from one base to another. This is necessary as the geometric matches will not necessarily contain the right sequence of bases. You should never need to change this. Remember to put the file name in *single* quotes.
searchStructureFile	Specifies the name of the PDB file containing the reference full atomic structure. Remember to put the file name in *single* quotes.
nhradius	Specifies a cutoff for searching for non-helical matches (in Angstroms). This is used to control the strictness of the match search. If one of your non-helical fragments results in zero matches, increase this number.

Parameter	**Details**
hradius	Specifies a cutoff for searching for helical matches (in Angstroms). This is used to control the strictness of the match search. If one of your helical fragments results in zero matches, increase this number.

3. Now let's run the script `ex1.py`. In the command prompt/terminal window, type:

   (Windows) `\Python26\python ex1.py`
   (Mac OS/Linux) `python ex1.py`

   You will see an output that looks like that shown in the figure below. The script outputs how long it took to find matching fragments for each fragment of the template coarse-grained model. A warning appears if less than 200 matches are found.



4. Verify that you have at least 1 match for each fragment. You can do this in one of two ways:

a. Scroll through the script output and see if any of the warnings say:

Warning, keeping only 0 options for ...

If you see such a warning, then no matches were found for that particular fragment and you should re-run the script with larger values for `nhradius` and/or `hradius`. If the line below the warning refers to a helix, increase the `hradius` value. Otherwise, increase the `nhradius` value. Try using increments of 1 Angstrom when increasing these variables.

b. Look at the output files generated by *ex1.py*. The output file names are of the format <*fragment-identifier*>*-stats.dat*. So in your command prompt/terminal window, type:

(Windows) `dir *-stats.dat`
(Mac OS/Linux) `ls -l *-stats.dat`

Look at the file sizes of each of these files. If any of them are 0, then no matches were found for that particular fragment and you should re-run the script with larger values for `nhradius` and/or `hradius`. If the name of the file contains the letter H followed by a number, the fragment with 0 matches is a helical fragment, therefore you should increase the `hradius` variable. Otherwise, it is a non-helical fragment, and you should increase the `nhradius` variable.

5. The *ex1.py* script also created a new file called *6TNA-C3-1N32-subA-0-lib.pkl*. The format of this output is:

"model name" – "reference molecule name" – "model frame id" –lib.pk

The "model name" comes from the `templateTrace` file name. The "reference molecule name" comes from the `fragmentFile` specification.

"Model frame id" identifies which frame the library is associated with. In this example, the `templateTrace` file only contained one frame (frame 0). However, it is possible to use a PDB file that contains more than one frame, in which case, multiple libraries would be generated, each with a different "model frame id," starting with 0.

This library is needed by the second step of C2A, which assembles the fragments into a full atomic structure.

4.5 Assembling fragments into a model

The second step in adding full atomic detail to a coarse-grained model is to assemble the matching fragments that were found.

1. Open a text editor to examine the file *ex2.py*. DO NOT use Microsoft Word or other similar programs to edit or create these files since they insert formatting instructions that are not readable by C2A.

```
#!/usr/bin/env python

__author__ = "Randall J. Radmer"
__version__ = "1.0"
__doc__ = """Script to run second part of c2a calcs (assemble molecules)."""


import sys, time
import simtk.nast.c2a_assembleOptions as ao

t0 = time.time()
ao.assemble(fragIn = '6TNA-FD.txt',
            coarseIn = '6TNA-C3.pdb',
            pieceLib = open('6TNA-C3-1N32-subA-0-lib.pkl'),
            outName = 'test',
            n = 1,
            cutoff = 0.5
           )

sys.stdout.write("Total run time: %.1f Sec\n" % (time.time()-t0))

~
~
~
~
```

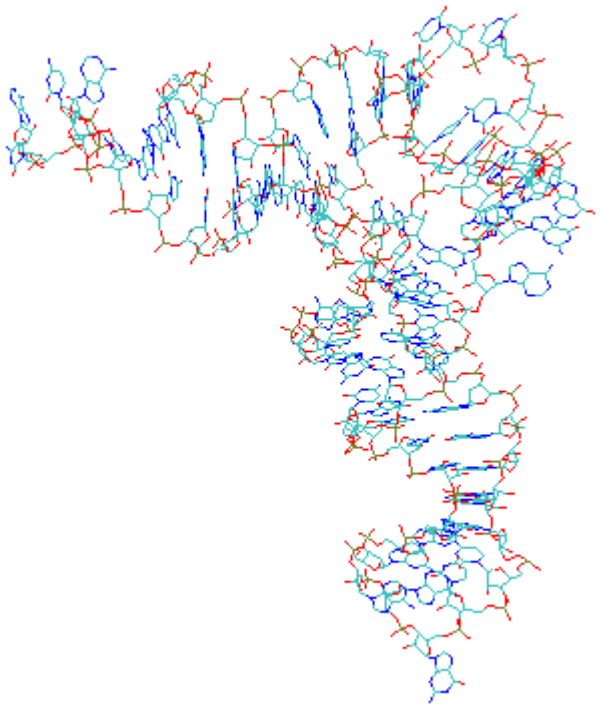
You should see a file like that shown above. This file is the script that assembles fragments from a working library into the full atomic structure. The table below describes each of the 6 parameters for the script in more detail.

Parameter	**Details**
fragmentFile	Specifies the name of the text file that defines the fragments of the coarse-grained model that need to be matched. See Appendix B for more details. Remember to put the file name in quotes.
coarseIn	Specifies the name of the PDB file containing the coarse-grained model. Remember to put the file name in quotes.
pieceLib	Specifies the name of the library that contains the matched fragments. This is one of the outputs from running the first step of C2A (using script *ex1.py*). Remember to put the file name in quotes.
outName	This is the root name for the output files. Remember to put the file name in quotes.
n	Specifies the number of full atomic structures to make
cutoff	Specifies the distance (in Angstroms) that defines a "collision." A fragment, if inserted into the full atomic model, would create a "collision" if it is closer to another atom than this cutoff distance and would not be used as part of the final model.

2. Now let's run the script `ex2.py`. In the command prompt/terminal window, type:

(Windows) `\Python26\python ex2.py`
(Mac OS/Linux) `python ex2.py`

The file *test-0.pdb* will be created. Load this file into VMD to visualize it (see detailed instructions under "Your first NAST run"). You may notice some gaps in the structure. This is the result of full atomic fragments coming from disjointed parts of the full atomic reference molecule, being assembled together in a new way. The gaps are large enough to make the structure chemically unrealistic; however, they can easily be minimized by using a classical molecular dynamics program, like GROMACS (OpenMM Zephyr provides an easy-to-use interface to GROMACS – see details in Section 4.7).



4.6 Running C2A on the coarse-grained model you generated with NAST

Now that you've learned the basis of running C2A, you can try it on the coarse-grained model that you generated earlier with NAST (Section 3.6)

1. Copy over the coarse-grained model you saved from the NAST-generated trajectory (*T4-last.pdb)*. In the command prompt/terminal window, type:

   (Windows) `copy ..\6TNA_MD\T4-last.pdb .`
   (Mac OS/Linux) `cp ../6TNA_MD/T4-last.pdb .`

2. Copy *ex1.py* to *myEx1.py*. You will do your edits to *myEx1.py*. Type:

   (Windows) `copy ex1.py myEx1.py`
   (Mac OS/Linux) `cp ex1.py myEx1.py`

3. Open *myEx1.py* in a text editor and modify it to use your coarse-grained model. So set:

   ```
   templateTrace = 'T4-last.pdb'
   ```

   No other parameters will change in this case. You will use the same fragment definition as 6TNA-C3, since we are still working with the same tRNA molecule, and the same reference molecule.

4. Now let's run your edited script `myEx1.py`. In the command prompt/terminal window, type:

   (Windows) `\Python26\python myEx1.py`
   (Mac OS/Linux) `python myEx1.py`

Check for warnings that 0 options were found. If this warning appears, re-run the script with larger values for `nhradius` and/or `hradius`.

Also verify that the *T4-last-1N32-subA-0-lib.pkl* file was created by listing the contents of the directory:

(Windows) `dir`
(Mac OS/Linux) `ls`

5. Now set up the files to run step 2 of C2A. Copy *ex2.py* to *myEx2.py*. You will do your edits to *myEx2.py*. Type:

(Windows) `copy ex2.py myEx2.py`
(Mac OS/Linux) `cp ex2.py myEx2.py`

6. Open *myEx2.py* in a text editor and modify it to use your coarse-grained model and the new working library. Also, change the output root name. So set:

```
coarseIn = 'T4-last.pdb'
pieceLib = open('T4-last-1N32-subA-0-lib.pkl')
outName='T4-FA'
```

7. Now let's run your edited script `myEx2.py`. In the command prompt/terminal window, type:

(Windows) `\Python26\python myEx2.py`
(Mac OS/Linux) `python myEx2.py`

8. Visualize the output file *T4-FA-0.pdb* within VMD.

9. You may wish to make more than one full atomic structure from your coarse-grained template by increase the value of `n`. Because of the random component in selecting and assembling matches, it is preferable to generate many full atomic structures (~5-10), as some may be better than others.

4.7 Modifying C2A-generated full atomic files for use with Amber96 force field

The PDB files generated by C2A are not compatible with the Amber96 force field, used within the molecular dynamics program GROMACS and OpenMM Zephyr, a program built on top of GROMACS. You can modify the output full atomic file to be used by these programs by using the *fixpdbforgromacs.py* script. For the previous exercise with a full atomic output file named *T4-FA-0.pdb*, you would type:

(Windows)

```
\Python26\python fixpdbforgromacs.py T4-FA-0.pdb T4-FA-0-mod.pdb
```

(Mac OS/Linux)

```
python fixpdbforgromacs.py T4-FA-0.pdb T4-FA-0-mod.pdb
```

Note that you provide the input file name, followed by the name for the new modified file.

You can later use the OpenMM Zephyr program with your *T4-FA-0-mod.pdb* file to minimize the chemically unrealistic gaps in the C2A output structure.

5 Generating Input Files for NAST and C2A

5.1 Overview

Both NAST and C2A require a number of input files. NAST requires input files for the primary sequence and for the secondary structure and optionally a file for the tertiary contacts. C2A requires a fragment definition file, describing the structure in terms of helices, loops, junctions and ends. These required files can all be generated automatically if you have a BPSEQ format.

In this chapter, you will learn about .bpseq files and how to generate the NAST and C2A files from them. Specific details about the input files themselves can be found in Appendices A (NAST input files) and B (C2A input file).

5.2 The BPSEQ file format

The BPSEQ file format is a simple text file that describes structural information. It contains one line for each base in the molecule. Each row contains three columns. The first column specifies the sequence position, starting at one. The second column lists the base using its one-letter notation. The third column lists the position number of the base with which is paired. If the base is unpaired, the third column is zero.

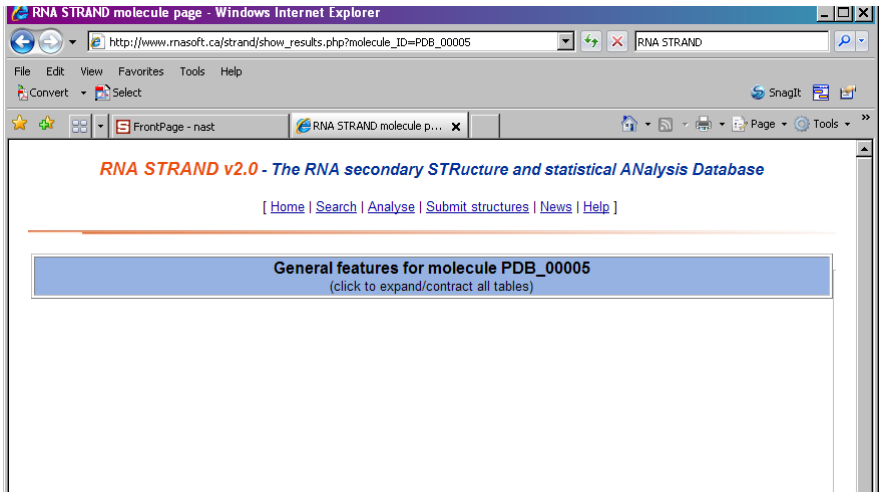
Structures in the BPSEQ file format can be obtained from websites, such as the Comparative RNA Web site and project (http://www.rna.ccbb.utexas.edu/DAT/) and RNA STRAND (http://www.rnasoft.ca/strand/).

5.3 Generating NAST and C2A files from BPSEQ files

The instructions below explain how to generate the initial files needed for NAST and C2A from a BPSEQ file. After you obtain these initial files, you can test your familiarity with the process by generating a coarse grain model from them and then adding in the full atomic detail.

1. Go to RNA STRAND (http://www.rnasoft.ca/strand/) and download the BPSEQ file for PDB_00005:

    a. Enter `PDB_00005` in the search box and click "Search RNA STRAND ID." You should be taken to a page that looks like that shown in the figure below.



    b. Click on the instructions "click to expand/contract all tables" near the top to bring up a table of information to the right of the webpage.

RNA STRAND v2.0 - The RNA secondary STRucture and statistical ANalysis Database

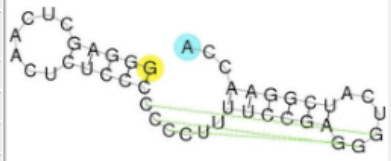
[Home | Search | Analyse | Submit structures | News | Help]

General features for molecule PDB_00005
(click to expand/contract all tables)

Format: CT (CT / RNAML / Bpseq / Dot-parentheses / FASTA) — View the RNA sequence and secondary structure for molecule PDB_00005

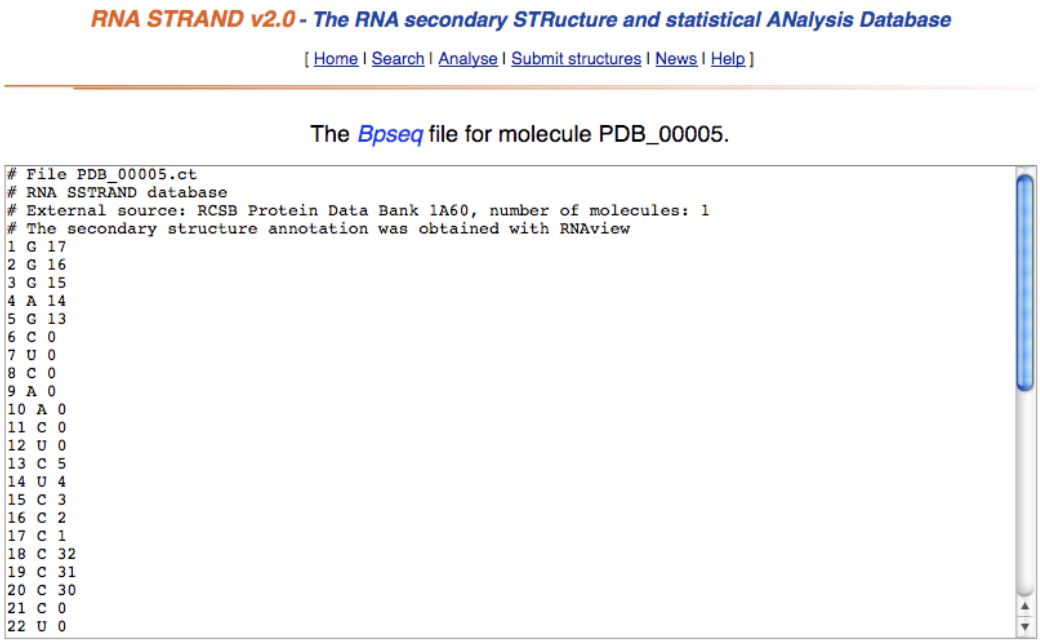
the RNA Secondary Structure Analyser for molecule PDB_00005

Molecule ID [?]:	PDB_00005
Molecule name [?]:	NMR STRUCTURE OF A CLASSICAL PSEUDOKNOT: INTERPLAY OF SINGLE-AND DOUBLE-STRANDED RNA, 24 STRUCTURES
Source [?]:	RCSB Protein Data Bank
Source ID [?]:	1A60
Reference [?]:	M.H.KOLK,M.VAN DER GRAAF,S.S.WIJMENGA,C.W.PLEIJ, H.A.HEUS,C.W.HILBERS. NMR STRUCTURE OF A CLASSICAL PSEUDOKNOT: INTERPLAY OF SINGLE- AND DOUBLE-STRANDED RNA.. SCIENCE V. 280 434 1998 ASTM SCIEAS US ISSN 0036-8075
Type [?]:	Other RNA
Organism [?]:	N/A
Validated by NMR or X-Ray [?]:	Yes
Method for secondary structure determination [?]:	NMR; ran through RNAview
Number of molecules [?]:	1
Length [?]:	44
Fragments used [?]:	Yes
Duplicated sequence [?]:	No other molecule in the database has the same sequence
Number of domains [?]:	2
Number of unpaired bases [?]:	16
Number of paired bases [?]:	28



[?] PS figure PDF figure
Figure from original source

c. Select “Bpseq” from the “Format” drop-down menu. Then click on “View the RNA sequence and secondary structure for molecule PDB_00005.” You will see a webpage that looks like that in the figure below.



d. Cut and paste the results into a file named PDB_00005.bpseq. The file name must have the extension .bpseq.

Windows users: If you are using WordPad, you will be required to specify a document type. Choose "Text Document." This will save the file in the needed format, but it will append .txt to the end of your file name. To remove the .txt, go to your command prompt window and navigate to the directory where you saved the file:

(Windows) `cd <directory>`

Then, copy the file to a new name ending with .bpseq. For example, if the WordPad file was saved as PDB_00005.bpseq.txt, then you would type:

(Windows) `copy PDB_00005.bpseq.txt PDB_00005.bpseq`

Mac OSX TextEdit users: If you are using TextEdit, you will first need to create a blank file from the terminal window with the .bpseq extension.

TextEdit is not able to create pure text files, although it can edit them. To create the blank text file, type:

(Mac OS) `touch PDB_00005.bpseq`

Open PDB_00005.bpseq in TextEdit and paste in the data. Save the file.

2. Create a new directory for PDB_00005 within the NAST examples folder and move your PDB_00005.bpseq file into it.
    a. Navigate back to the examples/nast folder. If you are continuing from the C2A exercises, this should just be one directory level higher so you would type:

    (Windows) `cd ..`
    (Mac OS / Linux) `cd ..`

    b. Make a new folder called *PDB_00005* (that's 4 zeros!)

    (Windows) `mkdir PDB_00005`
    (Mac OS / Linux) `mkdir PDB_00005`

    c. Put your *PDB_00005.bpseq* file into this new *PDB_00005* directory.

3. Go to the new directory from within your command prompt/terminal window:

(Windows) `cd PDB_00005`
(Mac OS / Linux) `cd PDB_00005`

4. In the new *PDB_00005* directory, generate the input files needed for NAST and C2A by typing:

(Windows) `\Python26\python ..\parseBPseq.py PDB_00005`
(Mac OS / Linux) `python ../parseBPseq.py PDB_00005`

In specifying the file name, you do *not* include the .bpseq extension.

After the script completes, you should have three new files in the directory: *PDB_00005-helix.txt*, *PDB_00005.seq*, and *PDB_00005-FD.txt*. To check the contents of your directory, type the following into your command prompt/terminal window:

(Windows) `dir`
(Mac OS / Linux) `ls`

The current version of parseBPseq.py cannot handle tertiary contacts in the .bpseq file (it will create a one-pair-long helix). You must specify tertiary contacts by writing your own tertiary contact file.

You can now use these files to build a coarse grain model and add full atomic detail. Try this out on your own!

6 Appendix A: NAST Input Files

NAST requires a primary sequence file and a secondary structure file. It can optionally take a tertiary contacts file. The format for all these files are described below.

6.1 Primary sequence file

File name format: Must NOT end with the extension .pdb, which is reserved for reading structures into NAST

Format:

- One residue per line
- Only options are: A, U, C, G

  (Technically, NAST can handle any type of residue, including modified residues. However, C2A can only handle the regular bases, which have to be capitalized – modified bases are often represented with lower case letters, like g. If you intend to use the output of NAST with C2A, you are limited to regular bases.)
- For a break in the sequence (e.g., more than one strand), use TER

Example:

```
A
G
C
U
TER
A
G
C
U
```

See also 6TNA_C3.seq (in the NAST examples folder under examples/nast/6TNA_MD/).

6.2 Secondary structure file

Format:

- Two lines per helix, one blank line separating helices
- The numbers represent residues based on their position in the sequence, with numbering starting at 1.
- Paired residues line up vertically

In the example 1 below, there are 2 helices. In the first, residue 1 lines up with 20; 2 with 19; 3 with 18; and 4 with 17. In the second helix, residue 6 lines up with 15; 7 with 14; and 8 with 13.

Example 1:

```
1 2 3 4
20 19 18 17

6 7 8
15 14 13
```

See also 6TNA_helix.txt (in the NAST examples folder under examples/nast/6TNA_MD/) . In this file a series of three periods (...) can be used to indicate a range, so the above example could have between written as follows:

Example 2:

```
1 … 4
20 … 17

6 … 8
15 … 13
```

This format is useful for allowing you to describe bulges, as shown in Example 3. Here, 1 pairs with 20; 19 is a bulge residue (not paired with anything); 2 pairs with 18 and so on.

Example 3:

```
1 2 … 4
20 18 … 16
```

6.3 Tertiary contacts

Format:

- One line per tertiary contact
- The numbers represent residues based on their position in the sequence, with numbering starting at 1.
- A tertiary contact is essentially a spring that connects the two specified residues. This interaction is specified in four columns:
    - First two columns are the two residues in the tertiary contact
    - Column 3 is the spring-preferred distance in nm
    - Last column is the spring strength in kJ/nm^2

In the example below, there are two tertiary contacts. The first row specifies the contact between residues 8 and 14. The spring-preferred distance is 1.3 nm and the spring strength is 200 kJ/nm^2. The distance value of 1.3 nm represents the approximate distance between the C3' atoms of two base-paired residues. The spring strength of 200 kJ/nm^2 has been determined to work best in our experiments. To constrain the desired distance more stringently (for example, if using known distances from crystal structures), increase this value (for example, try 500). If you want the tertiary contact distance to be more flexibly distributed around the

specified distance, decrease this value (for example, try 50). The second row specifies a similar contact between residues 15 and 48.

Example:

```
8   14   1.3  200
15  48   1.3  200
```

See also 6TNA_contacts.txt (in the NAST examples folder under examples/nast/6TNA_MD/) .

7 Appendix B: C2A Fragment Definition File

C2A requires a fragment definition file, which is described below.

File name format: Must end with *-FD.txt*

Example:

H1 1:7,66:72 10
H2 10:13,22:25 10
H3 27:31,39:43 10
H4 49:53,61:65 10
L1 H2 10
L2 H3 10
L3 H4 10
J1 H1:5,H2:5 10
J2 H2:3,H3:5 10
J3 H3:3,H4:5 10
E1 H1:3+4 10

Format:

- Helices:
    - Helix fragment definitions must start with the letter H
    - The first range of number (Residue1:Residue2) lists the residues that make up the 5' end of the helix

- The second range of numbers (Residue3:Residue4) lists the residues that make up the 3' end of the helix
- The last number specifies the number of choices to be used in the fragment assembly
- In the example above, the line `H1 1:7,66:72 10` instructs C2A that:
  H1 is the first helix.
  1:7 means that the 5' end of the helix is made up of residues 1 through 7.
  66:72 means the 3' end of the helix is made up of residues 66 to 72. That is 66 pairs with 7, etc.
  10 represents the number of choices to be used in the fragment assembly protocol. Here, the 10 best fragments will be the only choices for H1.

- Loops:
  - Loop definitions must start with the letter L
  - The middle entry specifies where the loop is located
  - The last value is the number of choices to be used in the fragment assembly protocol
  - In the example above, the line `L1 H2 10` instructs C2A that:
    L1 is the identifier for the loop.
    L1 is the loop at the end of H2.
    10 once again represents the number of choices for the assembly protocol.

- Junctions:
  - Junctions must start with the letter J
  - The second entry specifies what the 5' end of the junction is connected to
  - The third entry specifies what the 3' end of the junction is connected to
  - The last value is the number of choices to be used in the fragment assembly protocol

- In the example above, the line `J1 H1:5,H2:5 10` instructs C2A that:
  J1 is the identifier for this junction.
  H1:5 means that the junction is connected on the 5' end to the 5' half of helix H1.
  H2:5 means that the junction is connected on the 3' end to the 5' half of helix H2.
  10 represents the number of choices for the fragment assembly protocol.

- Ends:
  - Ends must start with the letter E
  - The second entry specifies where the end is connected to and in which direction it extends
  - The last entry is the number of choices to be used in the fragment assembly protocol
  - In the example above, the line `E1 H1:3+4 10` instructs C2A that:
    E1 is the indentifier.
    H1:3 means that the end is connected to the 3' half of helix H1.
    +4 means that the end extends from the helix by four residues counting up (if it were counting down, it would be -4)
    10 represents the number of choices for the fragment assembly protocol