User Manual for RNAJP

(version 1.0)

Shi-Jie Chen Research Group

The Chen research group investigates the physical mechanism of RNA folding and develops predictive models for RNA structures and functions.

Shi-Jie Chen
Principle Investigator
Curators' Distinguished Professor
Email: chenshi@missouri.edu
Group website: https://vfold.missouri.edu/chen-lab02.html

Mizzou
University of Missouri

Department of Physics and Astronomy
University of Missouri
Columbia, MO 65211
USA

Contents

1 Introduction

The RNAJP package is used for RNA 3D structure prediction from a given 2D structure. Its workflow is illustrated in Fig. 1. (1) Based on the given 2D structure, a circular coarse-grained 3D initial structure is generated. (2) From the circular structure, the non-pseudoknotted (non-PK) helices in the given 2D structure are folded by MD simulations. (3) If pseudoknotted (PK) helices exist in the given 2D structure, they are folded by MD simulations after the non-PK helices are folded in step (2). (4) After all the non-PK and PK helices are folded, an initial structure corresponding to the given 2D structure is obtained. (5) From the initial structure in step (4), simulated annealing MD simulations are performed repeatedly for conformational sampling by rapidly changing the junction topologies. (6) The low-energy structures in the sampled trajectory are clustered and the centroid coarse-grained structures in the clusters are extracted as the predicted structures. (7) The extracted coarse-grained structures are converted into the all-atom (AA) structures. (8) The all-atom structures are refined by the software QRNAS [1] to fix the broken bonds and remove the steric clashes.

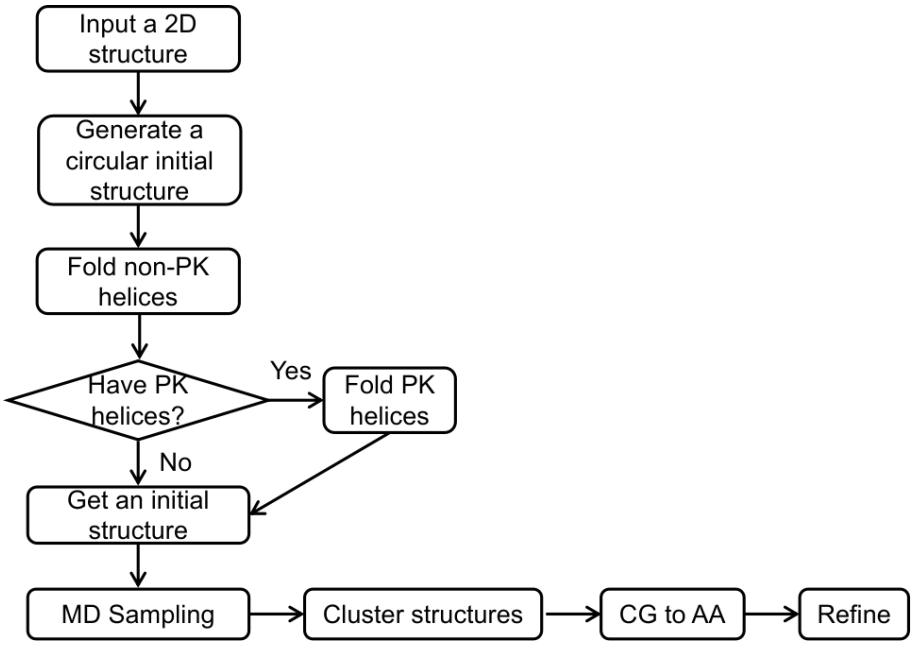

Figure 1: The workflow of the RNA 3D structure prediction procedures implemented in the RNAJP package.

2 User Guide

2.1 Overview of the RNAJP Package

The RNAJP package is written in Python and the MD simulations are implemented in the OpenMM package, a high-performance and flexible toolkit for molecular simulations. We will briefly introduce the python scripts and the sub-directories in the source code directory.

(1) scripts for the user interactive interface:

main.py is the main script to accept input information, such as 2D structure file, simulation time and so on. The detailed usage can be found in the “Usage for the RNAJP package" section below.

(2) scripts for parsing the given 2D structure:

parse2D.py is used to parse the given 2D structure, and return sequence, helix and loop information.

(3) scripts for generating the initial structures:

generate_circular_structure.py is used to generate a circular coarse-grained 3D structure based on the given 2D structure.

generate_initial_structure.py is used to run MD simulations to generate the initial structure where the non-PK and PK helices are folded.

fold_helices_from_circular_structure.py stores the energy functions for the MD simulations to fold the helices, which are imported by the above Python script “generate_initial_structure.py"

check_structure.py is used to check whether the helices in the given 2D structure are well folded in the generated initial structure, which is imported by the above Python script “generate_initial_structure.py". If the check is not passed, a new initial structure will be reproduced.

(4) scripts for conformational sampling:

fold_RNA.py is the script to run MD simulations for conformational sampling.

add_local_force.py stores the energy functions for the local bonds, angles and torsion.

add_non_local_force.py stores the energy functions for the non-local interactions, such as base pairing, base stacking, interactions between loops, and JAR3D interactions for the internal and hairpin loops.

add_junction_topology_force.py stores the energy functions for helix-helix correlation energy for the junction structures.

get_JAR3D_paras.py is used to extract the energy parameters for the JAR3D interactions by running the JAR3D model [2, 3] stored in the directory "JAR3D/".

add_constraint_force.py stores the energy functions for all kinds of constraints, such as the A-helix constraints, the distance/angle/torsion constraints, base pairing and stacking constraints.

(5) scripts for postprocessing:

postprocess.py is the script to postprocess the sampled trajectory, which includes calculating and reweighting the energy terms, clustering, converting CG structures into AA structures, and refining the AA structures.

recover_all_atom_structure.py is used to convert the CG structures into AA structures, which is imported by the above script “postprocess.py"".

(6) force field files:

myresidues.xml defines the new coarse-grained residue templates used in OpenMM.

myRNA.xml defines the new atom types, energy parameters for the new bonds ang angles used in OpenMM.

(7) The are five sub-directories in the source code directory:

angle_torsion_paras/ stores the energy parameters for the angles and torsion angles.

bp_stk_paras/ stores the energy parameters for the base pairing and stacking interactions.

JAR3D/ stores the external JAR3D model [2, 3] which is written in JAVA and the JAR3D energy parameters for the hairpin and internal loops.

nucleotide_templates_cg_to_aa/ stores the structural templates for the backbone and bases used for all-atom structure reconstruction.

QRNAS/ stores the external QRNAS model which is written in C++ and used for structural refinement [1].

(8) Moreover, we have provided a script named ***"run_cg_to_aa.py"*** to convert the coarse-grained structures into the all-atom structures. The detailed usage can be found in Example 4 in the "Examples" section below.

2.2 Installation

(1) Machine requirements

(a) Linux operating system

(b) $\geq$ 5 G RAM

(c) GPU

(d) g++ compiler

(e) Java version 8 (used for running the JAR3D model [2, 3])

(2) Prerequisites: the following software should be installed on the machine.

(a) **Anaconda** for Linux and Python version $\geq$ 3.8 via https://www.anaconda.com/. Suppose you have installed Anaconda in "path_to_Anaconda/".
MDAnalysis can be installed by "path_to_Anaconda/bin/pip install --upgrade MDAnalysis". It is used to read the simulation trajectory and calculate the RMSD.

(b) **OpenMM** (version $\geq$ 7.5) can be installed following the instruction on the OpenMM website OpenMM install instruction. It suggests installing OpenMM by "conda install -c conda-forge openmm". Users should use the "conda" command in the newly installed Anaconda. Then installation command should be "path_to_Anaconda/bin/conda install -c conda-forge openmm". After installation, users should test whether the installed OpenMM can use the GPUs.

(3) Installation of the RNAJP package

(a) tar -jxvf RNAJP_Linux.tar.bz2

(b) cd RNAJP_Linux/source/QRNAS and compile QRNAS by running command "make". If any errors, please see the installation instruction for QRNAS https://genesilico.pl/qrnas.

(c) Please set the environment variable "RNAJP_HOME" to the path where the source directory is.
export RNAJP_HOME=/Path_to_RNAJP_Linux/source
Please replace the path in the above line with the real path in your computer and add it to the file $\sim$/.bashrc. An absolute path should be used instead of a relative path.

2.3 Usage of the RNAJP package

The RNAJP package provides a Python interface to implement the 3D structure prediction from a given 2D structure.

The main script is `main.py` which is run by Python of version $\geq$ 3.8 and is stored in "RNAJP_Linux/source/". The Python command should be the one in the newly installed Anaconda. The main usage is as follows:

```
path_to_Anaconda/bin/python main.py -r rna -s 2D -t time -n npred
                                    -c constraint -g gpu
path_to_Anaconda/bin/python main.py -h
```

We will introduce the meaning of the above options.

```
-h                # Show this help message and exit
-r rna            # The name of RNA and working directory. default: test
-s 2D             # The file storing the secondary structure where the
                    first line is the sequence, and the second line is
                    the secondary structure in the dot-bracket format,
                    and different chains are separated by space.
-t time           # The simulation time in nanosecond (ns), 1 ns = 10^6
                    simulation steps. If not specified, a default time is
                    applied according to the given secondary structure.
-n npred          # The number of predicted structures. default: 10
-c constraint:    # The file containing the constraints. default: None
-g gpu            # The GPU index to be used if multiple GPUs are
                    available. default:0
```

There are five kinds of constraints available in the current RNAJP program, which are distance, angle, torsion, base stacking, and base pairing constraints. An example constraint file is as follows:

```
DIS A/18/N9 A/52/N9 3.0 6.0
  # distance constraint between two atoms A/18/N9 and A/52/N9 to make
    their distance in the range from 3.0 to 6.0 angstrom. Here 'A' stands
    for the chain name, 18 and 52 stand for the nucleotide index, and 'N9'
    stands for the atom name. The chain name starts from 'A', then 'B',
    'C' and so on. Limited by our CG representantion of a nucleotide, the
    accepted atom names are P, C4', (N9, C2, C6 for purines), (N1, C2, and
    C4 for pyrimidines).

ANGLE A/52/N9 A/52/C2 A/16/C2 50.0 90.0
  # angle constraint between three atoms A/52/N9, A/52/C2, and A/16/C2 to
    make them form an angle in the range from 50.0 to 90.0 degree.

TORSION A/52/N9 A/52/C6 A/16/C2 A/16/C4 0.0 10.0
  # torsion constraint between four atoms A/52/N9, A/52/C6, A/16/C2,
    and A/16/C4 to make them form a torsion angle in the range from 0.0 to
    10.0 degree.

STK A/52 A/18
  # base stacking constraint to make nucleotides A/52 and A/18 base
    stacked. 'A' is the chain name, and 52 and 18 are the nucleotide
    index.

BP A/16 A/52
  # base pairing constraint to make nucleotides A/16 and A/52 base paired.
```

2.4 Examples

We have provided four examples in the package. Users can change the input options for their prediction cases.

(A) Example 1: structure prediction for the RNA 1DK1 containing a 3-way junction.

This example can be run via the command “python ${RNAJP_HOME}/main.py -r 1dk1 -s 1dk1.2d -t 400 -n 10 -g 0". We have put this command in the file “run.sh”, and therefore you can run it via the command “bash run.sh”. Note: please use the “python” command in the newly installed Anaconda package (see the above installation). You can reduce the simulation time for a quick test, for example “-t 10” for 10 ns.

We will introduce the files to be generated.
(1) A working directory named “1dk1/" will be created immediately after running the above command. All the files to be generated will be stored in this directory.
(2) The file “circular_struc.pdb" stores the circular initial structure.
(3) The file “fold_helices_trj.pdb" stores the simulation trajectory to fold helices.
(3) The file “init.pdb" stores the initial structure after the helices in the given 2D structure are folded.
(4) The file “jar3d.pdb" stores the structure where the hairpin and internal loops are folded according to the structural templates extracted by the JAR3D model.
(5) The file “trj.pdb" stores the simulation trajectory for conformational sampling.
(6) The file “raw_energy.txt” records the energy terms for the structures in the trajectory file “trj.pdb".
(7) The file “reweighted_energy.txt" records the reweighted energies.
(8) The file “low_pot_trj.pdb” stores the low-energy structures.
(9) The file “low_pot_trj_energy.txt" stores the energies for the low-energy structures.
(10) THe file “low_pot_trj_pairwise_rmsd.txt" stores the pairwise RMSDs for the low-energy structures.
(11) The file “predicted_1dk1_cg-N.pdb” is the top-N predicted coarse-grained structure. N is from 1 to 10 for this case.
(12) The file “predicted_1dk1-N.pdb” is the top-N predicted all-atom structure rebuilt from the corresponding coarse-grained one. N is from 1 to 10 for this case.
(13) The file “em_predicted_1dk1-N.pdb” is the top-N predicted all-atom structure after energy minimization. N is from 1 to 10 for this case. These structures are the finally predicted structures.

We have provided the reference results in the directory “1dk1-Reference-Results/".

(B) Example 2: structure prediction for the pseudoknotted (PK) RNA 2BJ2 containing two chains.

This example can be run via the command “python ${RNAJP_HOME}/main.py -r 2bj2 -s 2bj2.2d -t 50 -n 10 -g 0". We have put this command in the file “run.sh”, and therefore you can run it via the command “bash run.sh”. Note: please use the “python” command in the newly installed Anaconda package (see the above installation). You can reduce the simulation time for a quick test, for example “-t 10” for 10 ns.

The files to be generated are almost the same as in Example 1, except for two additional files.
One file is “fold_helices_trj_nonPK.pdb" storing the trajectory folding the non-PK helices.
The other file is “init_nonPK.pdb" storing the structure where the non-PK helices are folded.
After folding the non-PK and PK helices, the initial structure is still stored in the file “init.pdb".

We have provided the reference results in the directory “2bj2-Reference-Results/".

(C) Example 3: structure prediction for the RNA 1DK1 with all kinds of constraints.

This example can be run via the command “python ${RNAJP_HOME}/main.py -r 1dk1 -s 1dk1.2d -t 50 -n 10 -g 0 -c constraints-1dk1.txt". We have put this command in the file “run.sh”, and therefore you can run it via the command “bash run.sh”. Note: please use the “python” command in the newly installed Anaconda package (see the above installation). You can reduce the simulation time for a quick test, for example “-t 10” for 10 ns.

There are seven constraints in the constraint file “constraints-1dk1.txt", which include the distance, angle, torsion, base pairing and base stacking constraints. The file format for the constraints are described in the above section “Usage of the RNAJP package".

The files to be generated are the same as in Example 1. We have provided the reference results in the directory “1dk1-constraints-Reference-Results/".

(D) Example 4: all-atom structure reconstruction

In this example, we will convert the two coarse-grained structures in the file “cg.pdb” to the all-atom structures which will be stored in the file “aa.pdb”. The command is as follows:

```
python ${RNAJP_HOME}/run_cg_to_aa.py cg.pdb aa.pdb
```

We have put this command in the file "run.sh" and thus you can run it via the command “bash run.sh". It takes about 10 seconds to reconstruct these two structures. But it will take some time if there are too many coarse-grained structures especially for large RNAs in the file “cg.pdb".

3 References

[1] Stasiewicz, J., Mukherjee, S., Nithin, C., and Bujnicki, J. M. (2019) QRNAS: software tool for refinement of nucleic acid structures. *BMC Struct. Biol.*, **19**, 1–11.

[2] Zirbel, C. L., Roll, J., Sweeney, B. A., Petrov, A. I., Pirrung, M., and Leontis, N. B. (2015) Identifying novel sequence variants of RNA 3D motifs. *Nucleic Acids Res.*, **43**, 7504–7520.

[3] Roll, J., Zirbel, C. L., Sweeney, B., Petrov, A. I., and Leontis, N. (2016) JAR3D Webserver: Scoring and aligning RNA loop sequences to known 3D motifs. *Nucleic Acids Res.*, **44**, W320–W327.