ScanFold3 User Guide

For questions, suggestions, or technical assistance, please contact:

Evelyn Coppenbarger, ecoppen@iastate.edu

Moss Lab Team, https://github.com/moss-lab

Please use the following citation for ScanFold3:

TBD

Table of Contents

1. Introduction

ScanFold is a program for finding, predicting, and scoring local RNA structures within long RNA or DNA sequences, such as entire genes or viral genomes. ScanFold uses a window-based approach to find short-ranged base pairs (default of 120 bases) which consistently reoccur across windows. The primary method of scoring base pairs via ScanFold is the thermodynamic z-score, which is how much more stable the structures a pair is found within are than would be expected from random chance. Pairs are assigned a z-score based on the normalized thermodynamic z-score of structures they occur within, which provides an indirect measurement of the stability that pair contributes to the structural ensemble. Covariation based analysis of ScanFold motifs is recommended, and along with 3D structure and bindable pocket prediction may be automated via the Cobretti pipeline, which can be found at https://github.com/moss-lab/Cobretti.

How to cite ScanFold3

TBD

How to cite Cobretti

Peterson JM, O'Leary CA, Coppenbarger EC, Tompkins VS, Moss WN. Discovery of RNA secondary structural motifs using sequence-ordered thermodynamic stability and comparative sequence analysis. MethodsX. 2023 Jun 29;11:102275. doi: 10.1016/j.mex.2023.102275. PMID: 37448951; PMCID: PMC10336498.

How to cite ScanFold2

Andrews RJ, Rouse WB, O'Leary CA, Booher NJ, Moss WN. ScanFold 2.0: a rapid approach for identifying potential structured RNA targets in genomes and transcriptomes. PeerJ. 2022 Nov 8;10:e14361. doi: 10.7717/peerj.14361. PMID: 36389431; PMCID: PMC9651051.

How to cite ScanFold1

Andrews RJ, Roche J, Moss WN. ScanFold: an approach for genome-wide discovery of local RNA structural elements-applications to Zika virus and HIV. PeerJ. 2018 Dec 18;6:e6136. doi: 10.7717/peerj.6136. PMID: 30627482; PMCID: PMC6317755.

How to cite RNAFold

R. Lorenz, S.H. Bernhart, C. Hoener zu Siederdissen, H. Tafer, C. Flamm, P.F. Stadler and I.L. Hofacker (2011), "ViennaRNA Package 2.0", Algorithms for Molecular Biology: 6:26

2. Background

Thermodynamic Z-Score

The primary metric used to interpret ScanFold results is an occurrence-normalized thermodynamic z-score[1]. A brief explanation is given here on how this score is calculated.

The thermodynamic z-score is a metric designed to interpret to what degree dinucleotide ordering impacts structural stability, which has been shown to perform more reliably than minimum free energy (MFE) alone[1]. A thermodynamic z-score is calculated for a given structure with a predicted (or measured) MFE by generating an ensemble of sequences large enough to have statistical significance (ScanFold uses 100), where each sequence is a copy of the input sequence that has been shuffled so as to preserve dinucleotide frequency[2]. Each of these sequences has a structure and associated MFE predicted using the Zuker Algorithm[3,4]. These MFEs are used to construct a normal curve, and a z-score (the number of standard deviations above/below the mean) is calculated for the input sequence. Negative z-scores are more likely than would be expected of a random ordering of those dinucleotides, suggesting higher order structures which influence stability.

It is important to note that ScanFold generates a z-score on a per-pair basis using the method described below. Unpaired nucleotides may have an associated z-score as well. This is calculated in the same way, and should be interpreted as the degree to which that nucleotide being paired disrupts structure, and provides information about the structural ensemble. A non-scored unpaired nucleotide is one that was simply out-competed for all possible pairs by other nucleotides, and is not as informative.

ScanFold Algorithm

The ScanFold algorithm consists of two steps. The first step, henceforth referred to as **Scan**, generates a list of folded windows generated from the input sequence, with an associated score for each window. The second step, henceforth referred to as **Fold**, generates a normalized z-score for each observed pairing (where unpaired nucleotides are represented as pairing to themselves)

Scan takes a windowed approach to calculating these scores. The input sequence is split up into a number of windows of a certain size (default 120 nucleotides), each of which has its first nucleotide separated by a given step size from the last (default 1, so that each nucleotide marks the beginning of a window until the window reaches the end of the input sequence). The sequence within these windows is folded to generate an MFE. In the original iteration of ScanFold[5], this sequence was then shuffled 100 times preserving dinucleotide frequency, with each shuffled sequence being refolded in order to generate a z-score for the original sequence as described above in the section **Thermodynamic Z-Score**. ScanFold2[6] introduced a machine learning algorithm trained to predict a mean and standard deviation for the normal curve of MFEs for a given sequence in order to speed up the Scan step of the algorithm, so folding is no longer explicitly performed. This algorithm was trained on sequences with lengths between 20 and 200 nucleotides, window lengths outside this range may give erroneous results.

ScanFold uses an optimized Zuker algorithm implemented in RNAFold[4] to predict structure and MFE for each window, which has a time complexity of $O(n^3)$ with respect to sequence length. This folding is done roughly once per nucleotide with the default step size of one. Since the sequence length is a fixed value the Scan algorithm technically scales to O(L) where L is the length of the input sequence, though this is scaled by a large constant factor: by default 120^3, or ~1.7 million, so in practice the scanning step is usually going to be the slower of the two except on very large input sequences.

Window-based data from Scan, stored as a tsv file in the root directory ScanFold was called from, is read by the Fold step. The Fold step constructs a base-pair matrix and stores summed data for the z-score and other metrics (see: Usage) as well as the number of times that pairing occurred. Unpaired nucleotides are represented as a nucleotide pairing to itself. To save space not all possible nucleotides are explicitly stored, instead only one windows length from the diagonal of the matrix is stored and attempts to access pairs outside of this will return an error.

When all data is read in, each pair has a coverage-normalized z-score calculated as described by Andrews et al[5], which is used as the basis for the rest of Fold and is considered the most informative metric. **Coverage-normalized z-scores are calculated using the following method:** the z-score of all windows a pair is stored within is summed together and divided by the number of windows the first nucleotide in a pair may have been found within. This will be either the window size (default 120) or the distance from the end of the sequence if within one window-length of the end.

Z-scores involving nucleotides within one windows length of the start or beginning of the sequence are less reliable, and it is recommended to either disregard these or double check with another algorithm or a smaller window length.

Structure determination

The base pair predictions made by ScanFold tend to be sparse, in that in general each nucleotide is usually only observed with a small number of possible pairs across window. However, most nucleotides still have some form of competition between different pairing targets (or being unpaired). To resolve this and give one optimal structure, Fold uses a greedy approximation algorithm to choose the best base pairs, pairing each nucleotide with at most one other target (or leaving it unpaired), from most to least negative until all bases have been paired or explicitly left unpaired. This was chosen as determining the minimum perfect weighted matching is computationally expensive, and a greedy approximation gives results very close to the minimum perfect weighted matching (as expected to the sparsity of graphs formed by observed pairs across windows).

Speed Considerations

The slowest step in resolving conflicting base pairs is sorting the list of all possible pairs. This is done using the std::sort() function included in the C++ standard library, which implements the Introsort algorithm[7]. Introsort chooses the most applicable of Quicksort[8], Heapsort[9], or Insertion Sort[10]. This has an average and worst case time complexity of O(mlogm) where m is the set of all scanned pairs, which is the theoretical time complexity of the Fold step. In practice formulating output may require parsing the set of all base pairs in the final structure multiple time and can add significant overhead. Another consideration is that while typically only a small subset of all possible pairs are found, meaning $m \approx L$ where m is all scanned pairs and L is the length of the input sequence, in the worst case scenario $m=Lw/2$, where w is window length (default 120). As this would require that each nucleotide can pair to half of all other nucleotides in its window (RNAFold under default settings only allows canonical and wobble pairs so a higher degree of pairing than this is not possible, no base can pair to another of its own type), and for sequence order to have little to no impact on possible pairing, it is unlikely to occur outside of circumstances such as very long stretches of AT repeats. We have observed in testing on the EBV genome that m is at most several times the size of L and that the time complexity of Fold is fairly consistent with respect to sequence length.

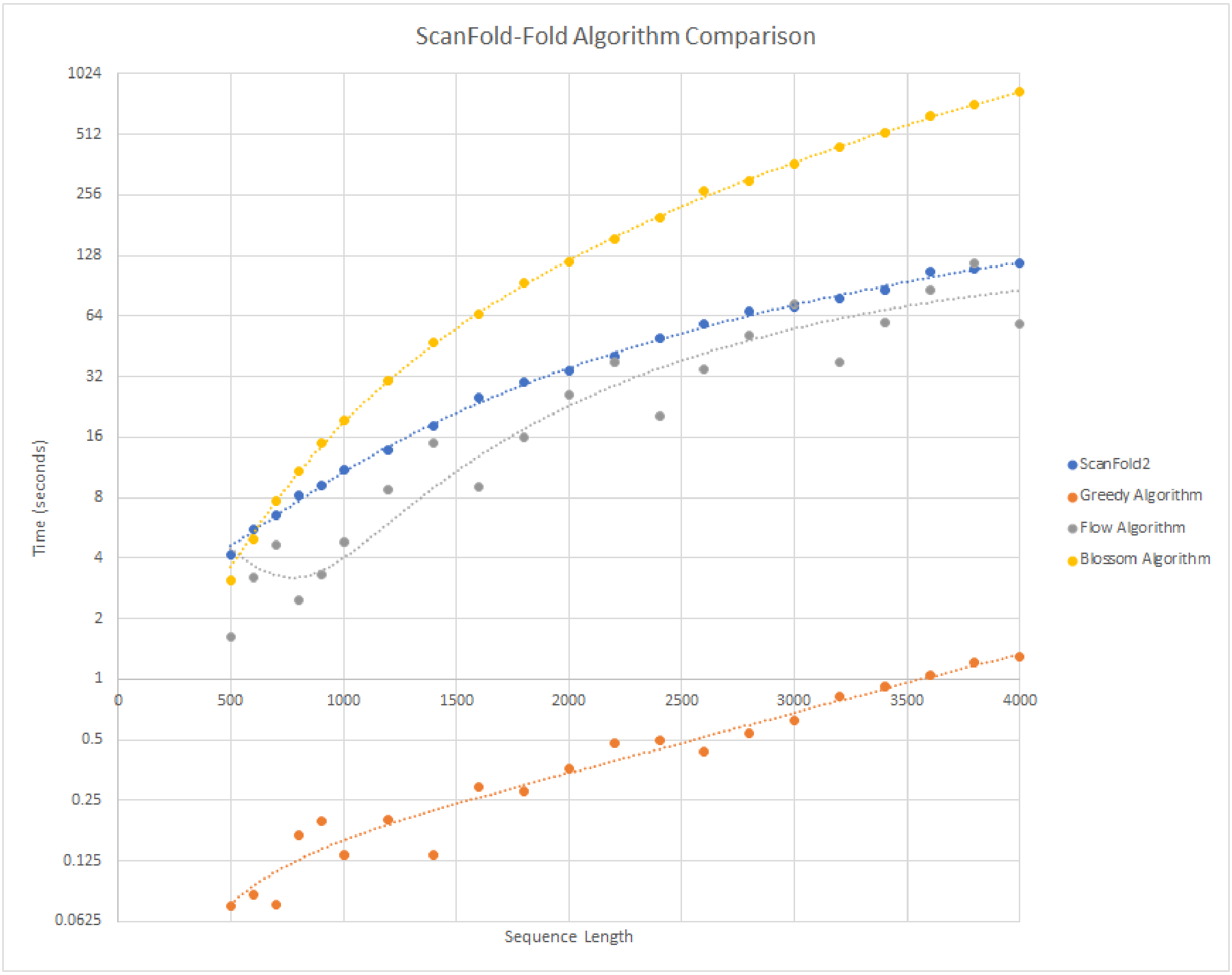

Figure 1: Benchmarking of the Fold step (algorithm only) for ScanFold2 (blue) and ScanFold3 (orange, gray, yellow). ScanFold3 was tested with 3 algorithms, the release version uses a greedy approximation algorithm (orange) due to large increase in speed while still finding a near-minimum matching. The Flow algorithm is a minimum-cost flow algorithm using the shorter path faster algorithm (SPFA). The Blossom algorithm is Edmonds' Blossom Algorithm, tested using the Boost C++ package. The dataset consisted sequences of the first n nucleotides of the Epstein-Barr Virus 1 reference genome, where n was a range of values from 500 to 4000. Testing was done for the entire genome but not shown here due to excessive memory usage in the Flow and Blossom algorithms on larger sequence lengths causing the program to fail.

3. Requirements

To run

Debian-based Operating System

Recommend Ubuntu 22+

Anaconda 29+

At least 7G of disc space

*ScanFold3 does NOT currently have Windows support, though it is planned for the full release. If necessary, for help getting ScanFold3 to work on Windows contact us over email or Github (see page 1 for contact information). We cannot assist in getting ScanFold3 to work on MacOS.

To compile (optional)

CMake 3.15+

PyBind11 2.13+

GCC Version 8 or later

4. Installation

ScanFold3 uses Conda to install dependencies. The Conda environment can be created using the following steps:

1. Download ScanFold3 from github (https://github.com/moss-lab/ScanFold3) to the desired folder

2. Open a terminal and navigate to the env directory in ScanFold3 and install the conda environment from ScanFold3.yml using the following command:

```
conda install -f ScanFold3.yml
```

3. When prompted, enter "y"

In order to run ScanFold3, the Conda environment must be active. This can be done using the command:

```
conda activate ScanFold3
```

This only needs to be done once per session. In order to run ScanFold3 in a new terminal you must activate the environment in it.

Compiling ScanFold3

ScanFold3 does not need to be compiled if you are using a compatible Debian-based operating system. If you are using a non-Debian OS or an old Debian OS, you may need to re-compile ScanFold3's libraries in order for it to work. If you encounter errors, particularly those caused by C++ name mangling (ImportError: undefined symbol: ...), re-compiling may fix them.

In order to compile ScanFold3, first ensure you have installed the requirements listed in *Requirements: To compile*. Then follow these steps:

1. In the ScanFold3 root directory, which contains the file CMakeLists.txt, create and navigate to a directory named "build":

```
mkdir build

cd build
```

2. In this directory, run the following commands:

```
cmake ..

make
```

3. Navigate back to the ScanFold3 directory. (Optional) delete the build directory:

```
cd ..

rm -r build
```

Troubleshooting Installation

If you are encountering errors running ScanFold3, try the following:

1. Ensure the Conda environment is active

2. Check Github for any updates

3. Double check your input files to ensure they are the proper format, with no empty lines or invisible characters. If the file was created in Windows, you may need to convert the file to Unix.

4. Re-compile ScanFold3's libraries (see *Installation: Compiling ScanFold3*)

5. Delete the ScanFold3 environment and re-create it (this may be necessary if you accidentally update the environment)

6. Check to ensure that the Conda environment is name correctly using the command:

```
conda env list
```

If it is not named ScanFold3, use the name listed here instead when activating the environment.

5. Usage

Running ScanFold

1. Ensure the conda environment created during installation is active. If using default installation, use this command:

```
conda activate ScanFold3
```

2. Run ScanFold using a fasta file input with the following command, with the path to where your fasta file and the ScanFold directory:

```
python /path/to/ScanFold.py your_sequence.fasta
```

ScanFold-Scan and ScanFold-Fold can optionally be run alone. ScanFold-Scan will produce only a .tsv file containing window data as an output. ScanFold-Fold must take a .tsv file in that format as an input.

To run ScanFold-Scan alone:

```
python /path/to/ScanFoldScan.py your_sequence.fasta
```

To run ScanFold-Fold alone:

```
python /path/to/ScanFoldFold.py your_sequence.fasta --tsv scan_output.tsv
```

In the case that you have multiple sequences in one fasta, one .tsv must be given for each sequence. Ensure headers in the fasta match with the name of the .tsv file given; everything before the first "." in the .tsv file name must be the same as the header for its respective sequence. Headers that contain a "." will cause an error.

Example (two sequence):

```
python /path/to/ScanFoldFold.py your_sequence.fasta --tsv header_1.tsv header_2.tsv
```

Flags (optional)

I/O

--folder_name

Name of output folder (defaults to date/time)

--extract

Extract structures from minus 1 or minus 2 dbn file (2 or 1); Default = 1

--tsv

Input tsv name from ScanFold-Scan (Only when directly running ScanFoldFold.py)

--id

Name or ID of sequence being analyzed (default "UserInput")

--es_path

Name of extracted structures file (default "extracted_structures")

--igv_path

Name of IGV file (default "igv_files")

--inforna_path

Name of inforna file (default "inforna_structures")

Webserver

These arguments are for use in the RNAStructuromeDB web server, primarily to display data through IGV. Most users will not need to use them.

--logfile

File where logs will be written (default stdout)

--loglevel

Logging level, see python logging library for more information on usage. (default "INFO")

--webserver

If provided, the output folder is compressed into a tar.gz file and written to the path specified by this parameter

--fasta_file_path

Web server specific fasta file path

--bp_track

Base pair track file, .bp format for use in webserver

--ed_wig_file_path

Ensemble diversity file, .wig format for use in webserver

--mfe_wig_file_path

Minimum free energy file, .wig format for use in webserver

--pvalue_wig_file_path

P-value file, .wig format for use in webserver

--zscore_wig_file_path

Thermodynamic z-score file, .wig format for use in webserver

--final_partners_wig

Base pairs after filtering for competition, .wig format for use in webserver

Scan Stage

-s, --step

Step size; default = 1

-w, --window

Window size; default = 120

Fold Stage

-f, --filter

Z-score value for filtering output, default = -1

-c, --competition

Competition determine if each base must pair to one and only one base, or if bases may be reported as pairing to multiple other bases (1 for disallow competition, 0 for allow; 1 by default)

--global_refold

Global refold option. Refold full sequence using Zavg <-1 and <-2 base pairs

Bibliography

1. Clote P, Ferré F, Kranakis E, Krizanc D. Structural RNA has lower folding energy than random RNA of the same dinucleotide frequency. RNA. 2005 May;11(5):578-91. doi: 10.1261/rna.7220505. PMID: 15840812; PMCID: PMC1370746.

2. Altschul SF, Erickson BW. Significance of nucleotide sequence alignments: a method for random sequence permutation that preserves dinucleotide and codon usage. Mol Biol Evol. 1985 Nov;2(6):526-38. doi: 10.1093/oxfordjournals.molbev.a040370. PMID: 3870875.

3. Lyngs R, Zuker M, Pedersen C. Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics. 1999;15(6):440. doi: 10.1093/bioinformatics/15.6.440.

4. Lorenz, R., Bernhart, S.H., Höner zu Siederdissen, C. et al. ViennaRNA Package 2.0. Algorithms Mol Biol 6, 26 (2011). https://doi.org/10.1186/1748-7188-6-26

5. Andrews RJ, Roche J, Moss WN. ScanFold: an approach for genome-wide discovery of local RNA structural elements-applications to Zika virus and HIV. PeerJ. 2018 Dec 18;6:e6136. doi: 10.7717/peerj.6136. PMID: 30627482; PMCID: PMC6317755.

6. Andrews RJ, Rouse WB, O'Leary CA, Booher NJ, Moss WN. ScanFold 2.0: a rapid approach for identifying potential structured RNA targets in genomes and transcriptomes. PeerJ. 2022 Nov 8;10:e14361. doi: 10.7717/peerj.14361. PMID: 36389431; PMCID: PMC9651051.

7. Musser DR. Introspective sorting and selection algorithms. Software: Practice and Experience. 1997 Aug;27(8):983-93.

8. Hoare CA. Quicksort. The computer journal. 1962 Jan 1;5(1):10-6.

9. Williams, J. W. J. (2025). Algorithm 232: Heapsort. Commun. ACM, 7(6), 347–348. https://doi.org/10.1145/512274.3734138

10. Kowalk, W.P. (2011). Insertion Sort. In: Vöcking, B., *et al.* Algorithms Unplugged. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15328-0_2