RNAlib-2.1.2 Reference Manual

Generated by Doxygen 1.4.7

Fri Jul 26 15:10:17 2013

Contents

Chapter 1

ViennaRNA Package core - RNAlib

A Library for folding and comparing RNA secondary structures

Date:

1994-2012

Authors:

Ivo Hofacker, Peter Stadler, Ronny Lorenz and many more

Table of Contents

1.1 Introduction

The core of the Vienna RNA Package (lorenz:2011, hofacker:1994) is formed by a collection of routines for the prediction and comparison of RNA secondary structures. These routines can be accessed through stand-alone programs, such as RNAfold, RNAdistance etc., which should be sufficient for most users. For those who wish to develop their own programs we provide a library which can be linked to your own code.

This document describes the library and will be primarily useful to programmers. However, it also contains details about the implementation that may be of interest to advanced users. The stand-alone programs are described in separate man pages. The latest version of the package including source code and html versions of the documentation can be found at

http://www.tbi.univie.ac.at/~ivo/RNA/

Chapter 2

RNAlib-2.1.2 Module Index

2.1 RNAlib-2.1.2 Modules

Here is a list of all modules:

Chapter 3

RNAlib-2.1.2 Data Structure Index

3.1 RNAlib-2.1.2 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

RNAlib-2.1.2 File Index

4.1 RNAlib-2.1.2 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

RNAlib-2.1.2 Page Index

5.1 RNAlib-2.1.2 Related Pages

Here is a list of all related documentation pages:

Chapter 6

RNAlib-2.1.2 Module Documentation

6.1 RNA Secondary Structure Folding

Collaboration diagram for RNA Secondary Structure Folding:This module contains all functions related to thermodynamic folding of RNAs.

Modules

- Calculating Minimum Free Energy (MFE) Structures
- Calculating Partition Functions and Pair Probabilities
- Enumerating Suboptimal Structures
- Calculate Secondary Structures of two RNAs upon Dimerization

  Predict structures formed by two molecules upon hybridization.

- Predicting Consensus Structures from Alignment(s)
- Predicting Locally stable structures of large sequences
- Change and Precalculate Energy Parameter Sets and Boltzmann Factors
- Energy evaluation

  This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

- Searching Sequences for Predefined Structures
- Classified Dynamic Programming
- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/fold.h

  This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

- Enumerating Suboptimal Structures

6.1.1 Detailed Description

This module contains all functions related to thermodynamic folding of RNAs.

6.2 Calculating Minimum Free Energy (MFE) Structures

Collaboration diagram for Calculating Minimum Free Energy (MFE) Structures:

Modules

- MFE Structures of two hybridized Sequences
- MFE Consensus Structures for Sequence Alignment(s)
- Local MFE structure Prediction and Z-scores
- Calculating MFE representatives of a Distance Based Partitioning

Functions

- float fold_par (const char $*$sequence, char $*$structure, paramT $*$parameters, int is_constrained, int is_circular)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

- float fold (const char $*$sequence, char $*$structure)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

- float circfold (const char $*$sequence, char $*$structure)

  Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.

- void free_arrays (void)

  Free arrays for mfe folding.

- void update_fold_params (void)

  Recalculate energy parameters.

6.2.1 Function Documentation

6.2.1.1 float fold_par (const char $*$ *sequence*, char $*$ *structure*, paramT $*$ *parameters*, int *is_constrained*, int *is_circular*)

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

The first parameter given, the RNA sequence, must be *uppercase* and should only contain an alphabet Σ that is understood by the RNAlib

(e.g. $\Sigma = \{A, U, C, G\}$)

The second parameter, *structure*, must always point to an allocated block of memory with a size of at least $\mathrm{strlen}(\mathrm{sequence}) + 1$

If the third parameter is NULL, global model detail settings are assumed for the folding recursions. Otherwise, the provided parameters are used.

The fourth parameter indicates whether a secondary structure constraint in enhanced dot-bracket notation is passed through the structure parameter or not. If so, the characters " | x $<$ $>$ " are recognized to mark bases that are paired, unpaired, paired upstream, or downstream, respectively. Matching brackets " ( ) " denote base pairs, dots "." are used for unconstrained bases.

To indicate that the RNA sequence is circular and thus has to be post-processed, set the last parameter to non-zero

After a successful call of fold_par(), a backtracked secondary structure (in dot-bracket notation) that exhibits the minimum of free energy will be written to the memory *structure* is pointing to. The function returns the minimum of free energy for any fold of the sequence given.

Note:

OpenMP: Passing NULL to the 'parameters' argument involves access to several global model detail variables and thus is not to be considered threadsafe

See also:

fold(), circfold(), model_detailsT, set_energy_model(), get_scaled_parameters()

Parameters:

sequence RNA sequence

structure A pointer to the character array where the secondary structure in dot-bracket notation will be written to

parameters A data structure containing the prescaled energy contributions and the model details. (NULL may be passed, see OpenMP notes above)

is_constrained Switch to indicate that a structure contraint is passed via the structure argument (0==off)

is_circular Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

Returns:

the minimum free energy (MFE) in kcal/mol

6.2.1.2 float fold (const char $*$ *sequence*, char $*$ *structure*)

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

This function essentially does the same thing as fold_par(). However, it takes its model details, i.e. temperature, dangles, tetra_loop, noGU, no_closingGU, fold_constrained, noLonelyPairs from the current global settings within the library

Use fold_par() for a completely threadsafe variant

See also:

fold_par(), circfold()

Parameters:

sequence RNA sequence

structure A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns:

the minimum free energy (MFE) in kcal/mol

6.2.1.3 float circfold (const char ∗ *sequence*, char ∗ *structure*)

Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.

This function essentially does the same thing as fold_par(). However, it takes its model details, i.e. temperature, dangles, tetra_loop, noGU, no_closingGU, fold_constrained, noLonelyPairs from the current global settings within the library

Use fold_par() for a completely threadsafe variant

See also:

fold_par(), circfold()

Parameters:

sequence RNA sequence

structure A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns:

the minimum free energy (MFE) in kcal/mol

6.3 Calculating Partition Functions and Pair Probabilities

Collaboration diagram for Calculating Partition Functions and Pair Probabilities:

Modules

- Compute the structure with maximum expected accuracy (MEA)
- Compute the centroid structure
- Partition Function for two hybridized Sequences
- Partition Function for two hybridized Sequences as a stepwise Process
- Partition Function and Base Pair Probabilities for Sequence Alignment(s)
- Partition functions for locally stable secondary structures
- Calculate Partition Functions of a Distance Based Partitioning

Functions

- float pf_fold_par (const char ∗sequence, char ∗structure, pf_paramT ∗parameters, int calculate_bppm, int is_constrained, int is_circular)

    Compute the partition function Q for a given RNA sequence.

- float pf_fold (const char ∗sequence, char ∗structure)

    Compute the partition function Q of an RNA sequence.

- float pf_circ_fold (const char ∗sequence, char ∗structure)

    Compute the partition function of a circular RNA sequence.

- void free_pf_arrays (void)

    Free arrays for the partition function recursions.

- void update_pf_params (int length)

    Recalculate energy parameters.

- void update_pf_params_par (int length, pf_paramT ∗parameters)

    Recalculate energy parameters.

- double ∗ export_bppm (void)

    Get a pointer to the base pair probability array
    Accessing the base pair probabilities for a pair (i,j) is achieved by.

- void assign_plist_from_pr (plist ∗∗pl, double ∗probs, int length, double cutoff)

    Create a plist from a probability matrix.

- int get_pf_arrays (short ∗∗S_p, short ∗∗S1_p, char ∗∗ptype_p, double ∗∗qb_p, double ∗∗qm_p, double ∗∗q1k_p, double ∗∗qln_p)

Get the pointers to (almost) all relavant computation arrays used in partition function computation.

- double mean_bp_distance (int length)

  Get the mean base pair distance of the last partition function computation.

- double mean_bp_distance_pr (int length, double ∗pr)

  Get the mean base pair distance in the thermodynamic ensemble.

6.3.1 Function Documentation

6.3.1.1 float pf_fold_par (const char ∗ *sequence*, char ∗ *structure*, pf_paramT ∗ *parameters*, int *calculate_bppm*, int *is_constrained*, int *is_circular*)

Compute the partition function Q for a given RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters " . , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If fold_constrained is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If tha parameter calculate_-bppm is set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise after calculations took place pr will contain the probability that bases *i* and *j* pair.

Note:

The global array pr is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function export_bppm()

Postcondition:

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable do_backtrack was set the base pair probabilities are already computed and may be accessed for further usage via the export_bppm() function. A call of free_pf_arrays() will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

See also:

pf_fold(), pf_circ_fold(), bppm_to_structure(), export_bppm(), get_boltzmann_-factors(), free_pf_arrays()

Parameters:

← ***sequence*** The RNA sequence input

- $\leftrightarrow$ ***structure*** A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)
- $\leftarrow$ ***parameters*** Data structure containing the precalculated Boltzmann factors
- $\leftarrow$ ***calculate_bppm*** Switch to Base pair probability calculations on/off (0==off)
- $\leftarrow$ ***is_constrained*** Switch to indicate that a structure contraint is passed via the structure argument (0==off)
- $\leftarrow$ ***is_circular*** Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

Returns:

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

6.3.1.2 float pf_fold (const char * *sequence*, char * *structure*)

Compute the partition function Q of an RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters " . , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If fold_constrained is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If do_backtrack has been set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise pr will contain the probability that bases *i* and *j* pair.

Note:

The global array pr is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function export_-bppm().
OpenMP: This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using pf_-fold_par() for a really threadsafe implementation.

Precondition:

This function takes its model details from the global variables provided in *RNAlib*

Postcondition:

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable do_backtrack was set the base pair probabilities are already computed and may be accessed for further usage via the export_bppm() function. A call of free_pf_arrays() will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

See also:

pf_fold_par(), pf_circ_fold(), bppm_to_structure(), export_bppm()

Parameters:

sequence The RNA sequence input

structure A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

Returns:

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

6.3.1.3 float pf_circ_fold (const char $*$ *sequence*, char $*$ *structure*)

Compute the partition function of a circular RNA sequence.

Note:

The global array pr is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function export_-bppm().
OpenMP: This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using pf_-fold_par() for a really threadsafe implementation.

Precondition:

This function takes its model details from the global variables provided in *RNAlib*

Postcondition:

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable do_backtrack was set the base pair probabilities are already computed and may be accessed for further usage via the export_bppm() function. A call of free_pf_arrays() will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

See also:

pf_fold_par(), pf_fold()

Parameters:

$\leftarrow$ ***sequence*** The RNA sequence input

$\leftrightarrow$ ***structure*** A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

Returns:

The Gibbs free energy of the ensemble ($G = -RT \cdot \log(Q)$) in kcal/mol

6.3.1.4 void free_pf_arrays (void)

Free arrays for the partition function recursions.

Call this function if you want to free all allocated memory associated with the partition function forward recursion.

Note:

Successive calls of pf_fold(), pf_circ_fold() already check if they should free any memory from a previous run.
OpenMP notice:
This function should be called before leaving a thread in order to avoid leaking memory

Postcondition:

All memory allocated by pf_fold_par(), pf_fold() or pf_circ_fold() will be free'd

See also:

pf_fold_par(), pf_fold(), pf_circ_fold()

6.3.1.5 void update_pf_params (int *length*)

Recalculate energy parameters.

Call this function to recalculate the pair matrix and energy parameters after a change in folding parameters like temperature

6.3.1.6 double* export_bppm (void)

Get a pointer to the base pair probability array

Accessing the base pair probabilities for a pair (i,j) is achieved by.

```
FLT_OR_DBL *pr  = export_bppm();
pr_ij           = pr[iindx[i]-j];
```

Precondition:

Call pf_fold_par(), pf_fold() or pf_circ_fold() first to fill the base pair probability array

See also:

pf_fold(), pf_circ_fold(), get_iindx()

Returns:

A pointer to the base pair probability array

6.3.1.7 void assign_plist_from_pr (plist $**$ *pl*, double $*$ *probs*, int *length*, double *cutoff*)

Create a plist from a probability matrix.

The probability matrix given is parsed and all pair probabilities above the given threshold are used to create an entry in the plist

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

Note:

This function is threadsafe

Parameters:

$\rightarrow$ ***pl*** A pointer to the plist that is to be created

$\leftarrow$ ***probs*** The probability matrix used for creting the plist

$\leftarrow$ ***length*** The length of the RNA sequence

$\leftarrow$ ***cutoff*** The cutoff value

6.3.1.8 int get_pf_arrays (short $**$ *S_p*, short $**$ *S1_p*, char $**$ *ptype_p*, double $**$ *qb_p*, double $**$ *qm_p*, double $**$ *q1k_p*, double $**$ *qln_p*)

Get the pointers to (almost) all relavant computation arrays used in partition function computation.

Precondition:

In order to assign meaningful pointers, you have to call pf_fold_par() or pf_fold() first!

See also:

pf_fold_par(), pf_fold(), pf_circ_fold()

Parameters:

$\rightarrow$ ***S_p*** A pointer to the 'S' array (integer representation of nucleotides)

$\rightarrow$ ***S1_p*** A pointer to the 'S1' array (2nd integer representation of nucleotides)

$\rightarrow$ ***ptype_p*** A pointer to the pair type matrix

$\rightarrow$ ***qb_p*** A pointer to the Q^B matrix

$\rightarrow$ ***qm_p*** A pointer to the Q^M matrix

$\rightarrow$ ***q1k_p*** A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1,k)$)

$\rightarrow$ ***qln_p*** A pointer to the 3' slice of the Q matrix ($qln(l) = Q(l,n)$)

Returns:

Non Zero if everything went fine, 0 otherwise

6.3.1.9 double mean_bp_distance (int *length*)

Get the mean base pair distance of the last partition function computation.

Note:

To ensure thread-safety, use the function mean_bp_distance_pr() instead!

See also:

mean_bp_distance_pr()

Parameters:

length

Returns:

mean base pair distance in thermodynamic ensemble

6.3.1.10 double mean_bp_distance_pr (int *length*, double $*$ *pr*)

Get the mean base pair distance in the thermodynamic ensemble.

This is a threadsafe implementation of mean_bp_dist() !

$< d >= \sum_{a,b} p_a p_b d(S_a, S_b)$

this can be computed from the pair probs $p_i j$ as

$< d >= \sum_{ij} p_{ij} (1 - p_{ij})$

Note:

This function is threadsafe

Parameters:

length The length of the sequence

pr The matrix containing the base pair probabilities

Returns:

The mean pair distance of the structure ensemble

6.4 Compute the structure with maximum expected accuracy (MEA)

Collaboration diagram for Compute the structure with maximum expected accuracy (MEA):

6.5 Compute the centroid structure

Collaboration diagram for Compute the centroid structure:

Functions

- char $*$ get_centroid_struct_pl (int length, double $*$dist, plist $*$pl)

  Get the centroid structure of the ensemble.

- char $*$ get_centroid_struct_pr (int length, double $*$dist, double $*$pr)

  Get the centroid structure of the ensemble.

6.5.1 Function Documentation

6.5.1.1 char$*$ get_centroid_struct_pl (int *length*, double $*$ *dist*, plist $*$ *pl*)

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for centroid() with a 'plist' input

The centroid is the structure with the minimal average distance to all other structures $<d(S)>=\sum_{(i,j)\in S}(1-p_{ij})+\sum_{(i,j)\notin S}p_{ij}$

Thus, the centroid is simply the structure containing all pairs with $p_ij > 0.5$ The distance of the centroid to the ensemble is written to the memory adressed by *dist*.

Parameters:

$\leftarrow$ ***length*** The length of the sequence

$\rightarrow$ ***dist*** A pointer to the distance variable where the centroid distance will be written to

$\leftarrow$ ***pl*** A pair list containing base pair probability information about the ensemble

Returns:

The centroid structure of the ensemble in dot-bracket notation

6.5.1.2 char$*$ get_centroid_struct_pr (int *length*, double $*$ *dist*, double $*$ *pr*)

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for centroid() with a probability array input

The centroid is the structure with the minimal average distance to all other structures $<d(S)>=\sum_{(i,j)\in S}(1-p_{ij})+\sum_{(i,j)\notin S}p_{ij}$

Thus, the centroid is simply the structure containing all pairs with $p_ij > 0.5$ The distance of the centroid to the ensemble is written to the memory adressed by *dist*.

Parameters:

$\leftarrow$ ***length*** The length of the sequence

$\rightarrow$ ***dist*** A pointer to the distance variable where the centroid distance will be written to

$\leftarrow$ ***pr*** A upper triangular matrix containing base pair probabilities (access via iindx get_iindx())

Returns:

The centroid structure of the ensemble in dot-bracket notation

6.6 Enumerating Suboptimal Structures

Collaboration diagram for Enumerating Suboptimal Structures:

Modules

- Suboptimal structures according to Zuker et al. 1989
- Suboptimal structures within an energy band arround the MFE
- Stochastic backtracking in the Ensemble

6.7 Suboptimal structures according to Zuker et al. 1989

Collaboration diagram for Suboptimal structures according to Zuker et al. 1989:

Functions

- SOLUTION ∗ zukersubopt (const char ∗string)

  Compute Zuker type suboptimal structures.

- SOLUTION ∗ zukersubopt_par (const char ∗string, paramT ∗parameters)

  Compute Zuker type suboptimal structures.

6.7.1 Function Documentation

6.7.1.1 SOLUTION∗ zukersubopt (const char ∗ *string*)

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker, i.e. for every possible base pair the minimum energy structure containing the resp. base pair. Returns a list of these structures and their energies.

Parameters:

string RNA sequence

Returns:

List of zuker suboptimal structures

6.8 Suboptimal structures within an energy band arround the MFE

Collaboration diagram for Suboptimal structures within an energy band arround the MFE:

Functions

- SOLUTION ∗ subopt (char ∗seq, char ∗structure, int delta, FILE ∗fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION ∗ subopt_par (char ∗seq, char ∗structure, paramT ∗parameters, int delta, int is_constrained, int is_circular, FILE ∗fp)

  Returns list of subopt structures or writes to fp.
- SOLUTION ∗ subopt_circ (char ∗seq, char ∗sequence, int delta, FILE ∗fp)

  Returns list of circular subopt structures or writes to fp.

Variables

- int subopt_sorted

  Sort output by energy.
- double print_energy

  printing threshold for use with logML

6.8.1 Function Documentation

6.8.1.1 SOLUTION∗ subopt (char ∗ *seq*, char ∗ *structure*, int *delta*, FILE ∗ *fp*)

Returns list of subopt structures or writes to fp.

This function produces **all** suboptimal secondary structures within 'delta' ∗ 0.01 kcal/mol of the optimum. The results are either directly written to a 'fp' (if 'fp' is not NULL), or (fp==NULL) returned in a SOLUTION ∗ list terminated by an entry were the 'structure' pointer is NULL.

Parameters:

seq

structure

delta

fp

Returns:

6.8.1.2 SOLUTION∗ subopt_circ (char ∗ *seq*, char ∗ *sequence*, int *delta*, FILE ∗ *fp*)

Returns list of circular subopt structures or writes to fp.

This function is similar to subopt() but calculates secondary structures assuming the RNA sequence to be circular instead of linear

Parameters:

seq

sequence

delta

fp

Returns:

6.9 Stochastic backtracking in the Ensemble

Collaboration diagram for Stochastic backtracking in the Ensemble:

Modules

- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)
- Stochastic Backtracking of Structures from Distance Based Partitioning

  Contains functions related to stochastic backtracking from a specified distance class.

Functions

- char ∗ pbacktrack (char ∗sequence)

  Sample a secondary structure from the Boltzmann ensemble according its probability

  .

- char ∗ pbacktrack_circ (char ∗sequence)

  Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

Variables

- int st_back

  Flag indicating that auxilary arrays are needed throughout the computations. This is essential for stochastic backtracking.

6.9.1 Function Documentation

6.9.1.1 char∗ pbacktrack (char ∗ *sequence*)

Sample a secondary structure from the Boltzmann ensemble according its probability

.

Precondition:

pf_fold_par() or pf_fold() have to be called first to fill the partition function matrices

Parameters:

sequence The RNA sequence

Returns:

A sampled secondary structure in dot-bracket notation

6.9.1.2 char∗ pbacktrack_circ (char ∗ *sequence*)

Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

This function does the same as pbacktrack() but assumes the RNA molecule to be circular

Precondition:

pf_fold_par() or pf_fold_circ() have to be called first to fill the partition function matrices

Parameters:

sequence The RNA sequence

Returns:

A sampled secondary structure in dot-bracket notation

6.9.2 Variable Documentation

6.9.2.1 int st_back

Flag indicating that auxilary arrays are needed throughout the computations. This is essential for stochastic backtracking.

Set this variable to 1 prior to a call of pf_fold() to ensure that all matrices needed for stochastic backtracking are filled in the forward recursions

See also:

pbacktrack(), pbacktrack_circ

6.10 Calculate Secondary Structures of two RNAs upon Dimerization

Collaboration diagram for Calculate Secondary Structures of two RNAs upon Dimerization:Predict structures formed by two molecules upon hybridization.

Modules

- MFE Structures of two hybridized Sequences
- Partition Function for two hybridized Sequences
- Partition Function for two hybridized Sequences as a stepwise Process

6.10.1 Detailed Description

Predict structures formed by two molecules upon hybridization.

The function of an RNA molecule often depends on its interaction with other RNAs. The following routines therefore allow to predict structures formed by two RNA molecules upon hybridization.

One approach to co-folding two RNAs consists of concatenating the two sequences and keeping track of the concatenation point in all energy evaluations. Correspondingly, many of the cofold() and co_pf_fold() routines below take one sequence string as argument and use the the global variable cut_point to mark the concatenation point. Note that while the *RNAcofold* program uses the '&' character to mark the chain break in its input, you should not use an '&' when using the library routines (set cut_point instead).

In a second approach to co-folding two RNAs, cofolding is seen as a stepwise process. In the first step the probability of an unpaired region is calculated and in a second step this probability of an unpaired region is multiplied with the probability of an interaction between the two RNAs. This approach is implemented for the interaction between a long target sequence and a short ligand RNA. Function pf_unstru() calculates the partition function over all unpaired regions in the input sequence. Function pf_interact(), which calculates the partition function over all possible interactions between two sequences, needs both sequence as separate strings as input.

6.11 MFE Structures of two hybridized Sequences

Collaboration diagram for MFE Structures of two hybridized Sequences:

Files

- file cofold.h

  MFE version of cofolding routines.

Functions

- float cofold (const char ∗sequence, char ∗structure)

  Compute the minimum free energy of two interacting RNA molecules.

- float cofold_par (const char ∗string, char ∗structure, paramT ∗parameters, int is_constrained)

  Compute the minimum free energy of two interacting RNA molecules.

- void free_co_arrays (void)

  Free memory occupied by cofold().

- void update_cofold_params (void)

  Recalculate parameters.

- void export_cofold_arrays_gq (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗fc_p, int ∗∗ggg_p, int ∗∗indx_p, char ∗∗ptype_p)

  Export the arrays of partition function cofold (with gquadruplex support).

- void export_cofold_arrays (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗fc_p, int ∗∗indx_p, char ∗∗ptype_p)

  Export the arrays of partition function cofold.

6.11.1 Function Documentation

6.11.1.1 float cofold (const char ∗ *sequence*, char ∗ *structure*)

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the fold() function. If cut_point ==-1 results should be the same as with fold().

Parameters:

sequence The two sequences concatenated

structure Will hold the barcket dot structure of the dimer molecule

Returns:

minimum free energy of the structure

6.11.1.2 void export_cofold_arrays_gq (int ∗∗ *f5_p*, int ∗∗ *c_p*, int ∗∗ *fML_p*, int ∗∗ *fM1_p*, int ∗∗ *fc_p*, int ∗∗ *ggg_p*, int ∗∗ *indx_p*, char ∗∗ *ptype_p*)

Export the arrays of partition function cofold (with gquadruplex support).

Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

Parameters:

f5_p A pointer to the 'f5' array, i.e. array conatining best free energy in interval [1,j]

c_p A pointer to the 'c' array, i.e. array containing best free energy in interval [i,j] given that i pairs with j

fML_p A pointer to the 'M' array, i.e. array containing best free energy in interval [i,j] for any multiloop segment with at least one stem

fM1_p A pointer to the 'M1' array, i.e. array containing best free energy in interval [i,j] for multiloop segment with exactly one stem

fc_p A pointer to the 'fc' array, i.e. array ...

ggg_p A pointer to the 'ggg' array, i.e. array containing best free energy of a gquadruplex delimited by [i,j]

indx_p A pointer to the indexing array used for accessing the energy matrices

ptype_p A pointer to the ptype array containing the base pair types for each possibility (i,j)

6.11.1.3 void export_cofold_arrays (int ∗∗ *f5_p*, int ∗∗ *c_p*, int ∗∗ *fML_p*, int ∗∗ *fM1_p*, int ∗∗ *fc_p*, int ∗∗ *indx_p*, char ∗∗ *ptype_p*)

Export the arrays of partition function cofold.

Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

Parameters:

f5_p A pointer to the 'f5' array, i.e. array conatining best free energy in interval [1,j]

c_p A pointer to the 'c' array, i.e. array containing best free energy in interval [i,j] given that i pairs with j

fML_p A pointer to the 'M' array, i.e. array containing best free energy in interval [i,j] for any multiloop segment with at least one stem

fM1_p A pointer to the 'M1' array, i.e. array containing best free energy in interval [i,j] for multiloop segment with exactly one stem

fc_p A pointer to the 'fc' array, i.e. array ...

indx_p A pointer to the indexing array used for accessing the energy matrices

ptype_p A pointer to the ptype array containing the base pair types for each possibility (i,j)

6.12 Partition Function for two hybridized Sequences

Collaboration diagram for Partition Function for two hybridized Sequences:

Files

- file part_func_co.h

  Partition function for two RNA sequences.

Functions

- cofoldF co_pf_fold (char ∗sequence, char ∗structure)

  Calculate partition function and base pair probabilities.

- cofoldF co_pf_fold_par (char ∗sequence, char ∗structure, pf_paramT ∗parameters, int calculate_bppm, int is_constrained)

  Calculate partition function and base pair probabilities.

- double ∗ export_co_bppm (void)

  Get a pointer to the base pair probability array.

- void free_co_pf_arrays (void)

  Free the memory occupied by co_pf_fold().

- void update_co_pf_params (int length)

  Recalculate energy parameters.

- void update_co_pf_params_par (int length, pf_paramT ∗parameters)

  Recalculate energy parameters.

- void compute_probabilities (double FAB, double FEA, double FEB, struct plist ∗prAB, struct plist ∗prA, struct plist ∗prB, int Alength)

  Compute Boltzmann probabilities of dimerization without homodimers.

- ConcEnt ∗ get_concentrations (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double ∗startconc)

  Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

Variables

- int mirnatog

  Toggles no intrabp in 2nd mol.

- double F_monomer [2]

  Free energies of the two monomers.

6.12.1 Function Documentation

6.12.1.1 cofoldF co_pf_fold (char ∗ *sequence*, char ∗ *structure*)

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the cut_-point nucleotide.

Note:

OpenMP: Since this function relies on the global parameters do_backtrack, dangles, temperature and pf_scale it is not threadsafe according to concurrent changes in these variables! Use co_pf_fold_par() instead to circumvent this issue.

See also:

co_pf_fold_par()

Parameters:

sequence	Concatenated RNA sequences
structure	Will hold the structure or constraints

Returns:

cofoldF structure containing a set of energies needed for concentration computations.

6.12.1.2 cofoldF co_pf_fold_par (char ∗ *sequence*, char ∗ *structure*, pf_paramT ∗ *parameters*, int *calculate_bppm*, int *is_constrained*)

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the cut_-point nucleotide.

See also:

get_boltzmann_factors(), co_pf_fold()

Parameters:

sequence	Concatenated RNA sequences
structure	Pointer to the structure constraint

parameters Data structure containing the precalculated Boltzmann factors

calculate_bppm Switch to turn Base pair probability calculations on/off (0==off)

is_constrained Switch to indicate that a structure contraint is passed via the structure argument (0==off)

Returns:

cofoldF structure containing a set of energies needed for concentration computations.

6.12.1.3 double∗ export_co_bppm (void)

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

See also:

get_iindx()

Returns:

A pointer to the base pair probability array

6.12.1.4 void update_co_pf_params (int *length*)

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings.

Note:

This function relies on the global variables pf_scale, dangles and temperature. Thus it might not be threadsafe in certain situations. Use update_co_pf_params_-par() instead.

See also:

get_boltzmann_factors(), update_co_pf_params_par()

Parameters:

length Length of the current RNA sequence

6.12.1.5 void update_co_pf_params_par (int *length*, pf_paramT ∗ *parameters*)

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings. It's second argument can either be NULL or a data structure containing the precomputed Boltzmann factors. In the first scenario, the necessary data structure will be created automatically according to the current global model settings, i.e. this mode might not be threadsafe. However, if the provided data structure is not NULL, threadsafety for the model parameters dangles, pf_scale and temperature is regained, since their values are taken from this data structure during subsequent calculations.

See also:

get_boltzmann_factors(), update_co_pf_params()

Parameters:

length Length of the current RNA sequence

parameters data structure containing the precomputed Boltzmann factors

6.12.1.6 void compute_probabilities (double *FAB*, double *FEA*, double *FEB*, struct plist ∗ *prAB*, struct plist ∗ *prA*, struct plist ∗ *prB*, int *Alength*)

Compute Boltzmann probabilities of dimerization without homodimers.

Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by assign_plist_from_pr(), the dimer probabilities 'prAB' are modified in place.

Parameters:

FAB free energy of dimer AB

FEA free energy of monomer A

FEB free energy of monomer B

prAB pair probabilities for dimer

prA pair probabilities monomer

prB pair probabilities monomer

Alength Length of molecule A

6.12.1.7 ConcEnt∗ get_concentrations (double *FEAB*, double *FEAA*, double *FEBB*, double *FEA*, double *FEB*, double ∗ *startconc*)

Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

This function takes an array 'startconc' of input concentrations with alternating entries for the initial concentrations of molecules A and B (terminated by two zeroes), then computes the resulting equilibrium concentrations from the free energies for the dimers. Dimer free energies should be the dimer-only free energies, i.e. the FcAB entries from the cofoldF struct.

Parameters:

FEAB Free energy of AB dimer (FcAB entry)

FEAA Free energy of AA dimer (FcAB entry)

FEBB Free energy of BB dimer (FcAB entry)

FEA Free energy of monomer A

FEB Free energy of monomer B

startconc List of start concentrations [a0],[b0],[a1],[b1],...,[an][bn],[0],[0]

Returns:

ConcEnt array containing the equilibrium energies and start concentrations

6.13 Partition Function for two hybridized Sequences as a stepwise Process

Collaboration diagram for Partition Function for two hybridized Sequences as a stepwise Process:

Files

- file part_func_up.h

    Partition Function Cofolding as stepwise process.

Functions

- pu_contrib ∗ pf_unstru (char ∗sequence, int max_w)

    Calculate the partition function over all unpaired regions of a maximal length.

- interact ∗ pf_interact (const char ∗s1, const char ∗s2, pu_contrib ∗p_c, pu_contrib ∗p_c2, int max_w, char ∗cstruc, int incr3, int incr5)

    Calculates the probability of a local interaction between two sequences.

- void free_interact (interact ∗pin)

    Frees the output of function pf_interact().

- void free_pu_contrib_struct (pu_contrib ∗pu)

    Frees the output of function pf_unstru().

6.13.1 Function Documentation

6.13.1.1 pu_contrib∗ pf_unstru (char ∗ *sequence*, int *max_w*)

Calculate the partition function over all unpaired regions of a maximal length.

You have to call function pf_fold() providing the same sequence before calling pf_unstru(). If you want to calculate unpaired regions for a constrained structure, set variable 'structure' in function 'pf_fold()' to the constrain string. It returns a pu_contrib struct containing four arrays of dimension [i = 1 to length(sequence)][j = 0 to u-1] containing all possible contributions to the probabilities of unpaired regions of maximum length u. Each array in pu_contrib contains one of the contributions to the total probability of being unpaired: The probability of being unpaired within an exterior loop is in array pu_contrib->E, the probability of being unpaired within a hairpin loop is in array pu_contrib->H, the probability of being unpaired within an interior loop is in array pu_contrib->I and probability of being unpaired within a multi-loop is in array pu_contrib->M. The total probability of being unpaired is the sum of the four arrays of pu_contrib.

This function frees everything allocated automatically. To free the output structure call free_pu_contrib().

Parameters:

sequence

max_w

Returns:

6.13.1.2 interact∗ pf_interact (const char ∗ *s1*, const char ∗ *s2*, pu_contrib ∗ *p_c*, pu_contrib ∗ *p_c2*, int *max_w*, char ∗ *cstruc*, int *incr3*, int *incr5*)

Calculates the probability of a local interaction between two sequences.

The function considers the probability that the region of interaction is unpaired within 's1' and 's2'. The longer sequence has to be given as 's1'. The shorter sequence has to be given as 's2'. Function pf_unstru() has to be called for 's1' and 's2', where the probabilities of being unpaired have to be given in 'p_c' and 'p_c2', respectively. If you do not want to include the probabilities of being unpaired for 's2' set 'p_c2' to NULL. If variable 'cstruc' is not NULL, constrained folding is done: The available constrains for intermolecular interaction are: '.' (no constrain), 'x' (the base has no intermolecular interaction) and '|' (the corresponding base has to be paired intermolecularily).

The parameter 'w' determines the maximal length of the interaction. The parameters 'incr5' and 'incr3' allows inclusion of unpaired residues left ('incr5') and right ('incr3') of the region of interaction in 's1'. If the 'incr' options are used, function pf_unstru() has to be called with w=w+incr5+incr3 for the longer sequence 's1'.

It returns a structure of type interact which contains the probability of the best local interaction including residue i in Pi and the minimum free energy in Gi, where i is the position in sequence 's1'. The member Gikjl of structure interact is the best interaction between region [k,i] k<i in longer sequence 's1' and region [j,l] j<l in 's2'. Gikjl_wo is Gikjl without the probability of beeing unpaired.

Use free_interact() to free the returned structure, all other stuff is freed inside pf_-interact().

Parameters:

s1

s2

p_c

p_c2

max_w

cstruc

incr3

incr5

Returns:

6.14 Predicting Consensus Structures from Alignment(s)

Collaboration diagram for Predicting Consensus Structures from Alignment(s):

Modules

- MFE Consensus Structures for Sequence Alignment(s)
- Partition Function and Base Pair Probabilities for Sequence Alignment(s)
- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)
- Local MFE consensus structures for Sequence Alignments

Functions

- int get_mpi (char ∗Alseq[ ], int n_seq, int length, int ∗mini)

  Get the mean pairwise identity in steps from ?to?(ident).

- float ∗∗ readribosum (char ∗name)

  Read a ribosum or other user-defined scoring matrix.

- float energy_of_alistruct (const char ∗∗sequences, const char ∗structure, int n_-seq, float ∗energy)

  Calculate the free energy of a consensus structure given a set of aligned sequences.

- void encode_ali_sequence (const char ∗sequence, short ∗S, short ∗s5, short ∗s3, char ∗ss, unsigned short ∗as, int circ)

  Get arrays with encoded sequence of the alignment.

- void alloc_sequence_arrays (const char ∗∗sequences, short ∗∗∗S, short ∗∗∗S5, short ∗∗∗S3, unsigned short ∗∗∗a2s, char ∗∗∗Ss, int circ)

  Allocate memory for sequence array used to deal with aligned sequences.

- void free_sequence_arrays (unsigned int n_seq, short ∗∗∗S, short ∗∗∗S5, short ∗∗∗S3, unsigned short ∗∗∗a2s, char ∗∗∗Ss)

  Free the memory of the sequence arrays used to deal with aligned sequences.

- int get_alipf_arrays (short ∗∗∗S_p, short ∗∗∗S5_p, short ∗∗∗S3_p, unsigned short ∗∗∗a2s_p, char ∗∗∗Ss_p, double ∗∗qb_p, double ∗∗qm_p, double ∗∗q1k_p, double ∗∗qln_p, short ∗∗pscore)

  Get pointers to (almost) all relavant arrays used in alifold's partition function computation.

Variables

- double cv_fact

  This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.

- double nc_fact

  This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

6.14.1 Function Documentation

6.14.1.1 int get_mpi (char * *Alseq*[ ], int *n_seq*, int *length*, int * *mini*)

Get the mean pairwise identity in steps from ?to?(ident).

Parameters:

Alseq

n_seq The number of sequences in the alignment

length The length of the alignment

mini

Returns:

The mean pairwise identity

6.14.1.2 float energy_of_alistruct (const char ** *sequences*, const char * *structure*, int *n_seq*, float * *energy*)

Calculate the free energy of a consensus structure given a set of aligned sequences.

Parameters:

sequences The NULL terminated array of sequences

structure The consensus structure

n_seq The number of sequences in the alignment

energy A pointer to an array of at least two floats that will hold the free energies (energy[0] will contain the free energy, energy[1] will be filled with the covariance energy term)

Returns:

free energy in kcal/mol

6.14.1.3 void encode_ali_sequence (const char ∗ *sequence*, short ∗ *S*, short ∗ *s5*, short ∗ *s3*, char ∗ *ss*, unsigned short ∗ *as*, int *circ*)

Get arrays with encoded sequence of the alignment.

this function assumes that in S, S5, s3, ss and as enough space is already allocated (size must be at least sequence length+2)

Parameters:

sequence The gapped sequence from the alignment

S pointer to an array that holds encoded sequence

s5 pointer to an array that holds the next base 5' of alignment position i

s3 pointer to an array that holds the next base 3' of alignment position i

ss

as

circ assume the molecules to be circular instead of linear (circ=0)

6.14.1.4 void alloc_sequence_arrays (const char ∗∗ *sequences*, short ∗∗∗ *S*, short ∗∗∗ *S5*, short ∗∗∗ *S3*, unsigned short ∗∗∗ *a2s*, char ∗∗∗ *Ss*, int *circ*)

Allocate memory for sequence array used to deal with aligned sequences.

Note that these arrays will also be initialized according to the sequence alignment given

See also:

free_sequence_arrays()

Parameters:

sequences The aligned sequences

S A pointer to the array of encoded sequences

S5 A pointer to the array that contains the next 5' nucleotide of a sequence position

S3 A pointer to the array that contains the next 3' nucleotide of a sequence position

a2s A pointer to the array that contains the alignment to sequence position mapping

Ss A pointer to the array that contains the ungapped sequence

circ assume the molecules to be circular instead of linear (circ=0)

6.14.1.5 void free_sequence_arrays (unsigned int *n_seq*, short $***$ *S*, short $***$ *S5*, short $***$ *S3*, unsigned short $***$ *a2s*, char $***$ *Ss*)

Free the memory of the sequence arrays used to deal with aligned sequences.

This function frees the memory previously allocated with alloc_sequence_arrays()

See also:

alloc_sequence_arrays()

Parameters:

n_seq The number of aligned sequences

S A pointer to the array of encoded sequences

S5 A pointer to the array that contains the next 5' nucleotide of a sequence position

S3 A pointer to the array that contains the next 3' nucleotide of a sequence position

a2s A pointer to the array that contains the alignment to sequence position mapping

Ss A pointer to the array that contains the ungapped sequence

6.14.1.6 int get_alipf_arrays (short $***$ *S_p*, short $***$ *S5_p*, short $***$ *S3_p*, unsigned short $***$ *a2s_p*, char $***$ *Ss_p*, double $**$ *qb_p*, double $**$ *qm_p*, double $**$ *q1k_p*, double $**$ *qln_p*, short $**$ *pscore*)

Get pointers to (almost) all relavant arrays used in alifold's partition function computation.

Note:

To obtain meaningful pointers, call alipf_fold first!

See also:

pf_alifold(), alipf_circ_fold()

Parameters:

S_p A pointer to the 'S' array (integer representation of nucleotides)

S5_p A pointer to the 'S5' array

S3_p A pointer to the 'S3' array

a2s_p A pointer to the pair type matrix

Ss_p A pointer to the 'Ss' array

qb_p A pointer to the Q^B matrix

qm_p A pointer to the Q^M matrix

q1k_p A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1,k)$)

qln_p A pointer to the 3' slice of the Q matrix ($qln(l) = Q(l,n)$)

Returns:

Non Zero if everything went fine, 0 otherwise

6.14.2 Variable Documentation

6.14.2.1 double cv_fact

This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.

Default is 1.

6.14.2.2 double nc_fact

This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

Default is 1.

6.15 MFE Consensus Structures for Sequence Alignment(s)

Collaboration diagram for MFE Consensus Structures for Sequence Alignment(s):

Functions

- float alifold (const char ∗∗strings, char ∗structure)

  Compute MFE and according consensus structure of an alignment of sequences.

- float circalifold (const char ∗∗strings, char ∗structure)

  Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.

- void free_alifold_arrays (void)

  Free the memory occupied by MFE alifold functions.

6.15.1 Function Documentation

6.15.1.1 float alifold (const char ∗∗ *strings*, char ∗ *structure*)

Compute MFE and according consensus structure of an alignment of sequences.

This function predicts the consensus structure for the aligned 'sequences' and returns the minimum free energy; the mfe structure in bracket notation is returned in 'structure'.

Sufficient space must be allocated for 'structure' before calling alifold().

Parameters:

strings A pointer to a NULL terminated array of character arrays

structure A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

Returns:

The free energy score in kcal/mol

6.15.1.2 float circalifold (const char ∗∗ *strings*, char ∗ *structure*)

Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.

Parameters:

strings A pointer to a NULL terminated array of character arrays

structure A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

Returns:

The free energy score in kcal/mol

6.16 Partition Function and Base Pair Probabilities for Sequence Alignment(s)

Collaboration diagram for Partition Function and Base Pair Probabilities for Sequence Alignment(s):

Functions

- float alipf_fold_par (const char ∗∗sequences, char ∗structure, plist ∗∗pl, pf_paramT ∗parameters, int calculate_bppm, int is_constrained, int is_circular)
- float alipf_fold (const char ∗∗sequences, char ∗structure, plist ∗∗pl)

  The partition function version of alifold() works in analogy to pf_fold(). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of pair_info structs. The list is terminated by the first entry with pi.i = 0.

- float alipf_circ_fold (const char ∗∗sequences, char ∗structure, plist ∗∗pl)
- double ∗ export_ali_bppm (void)

  Get a pointer to the base pair probability array.

6.16.1 Function Documentation

6.16.1.1 float alipf_fold_par (const char ∗∗ *sequences*, char ∗ *structure*, plist ∗∗ *pl*, pf_paramT ∗ *parameters*, int *calculate_bppm*, int *is_constrained*, int *is_circular*)

Parameters:

sequences

structure

pl

parameters

calculate_bppm

is_constrained

is_circular

Returns:

6.16.1.2 float alipf_fold (const char ∗∗ *sequences*, char ∗ *structure*, plist ∗∗ *pl*)

The partition function version of alifold() works in analogy to pf_fold(). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of pair_info structs. The list is terminated by the first entry with pi.i = 0.

Parameters:

sequences

structure

pl

Returns:

6.16.1.3 float alipf_circ_fold (const char ∗∗ *sequences*, char ∗ *structure*, plist ∗∗ *pl*)

Parameters:

sequences

structure

pl

Returns:

6.16.1.4 double∗ export_ali_bppm (void)

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

See also:

get_iindx()

Returns:

A pointer to the base pair probability array

6.17 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)

Collaboration diagram for Stochastic Backtracking of Consensus Structures from Sequence Alignment(s):

Functions

- char ∗ alipbacktrack (double ∗prob)

  Sample a consensus secondary structure from the Boltzmann ensemble according its probability

  .

6.17.1 Function Documentation

6.17.1.1 char∗ alipbacktrack (double ∗ *prob*)

Sample a consensus secondary structure from the Boltzmann ensemble according its probability

.

Parameters:

prob to be described (berni)

Returns:

A sampled consensus secondary structure in dot-bracket notation

6.18 Predicting Locally stable structures of large sequences

Collaboration diagram for Predicting Locally stable structures of large sequences:

Modules

- Local MFE structure Prediction and Z-scores
- Partition functions for locally stable secondary structures
- Local MFE consensus structures for Sequence Alignments

6.19 Local MFE structure Prediction and Z-scores

Collaboration diagram for Local MFE structure Prediction and Z-scores:

Functions

- float Lfold (const char ∗string, char ∗structure, int maxdist)

  The local analog to fold().

- float Lfoldz (const char ∗string, char ∗structure, int maxdist, int zsc, double min_z)

6.19.1 Function Documentation

6.19.1.1 float Lfold (const char ∗ *string*, char ∗ *structure*, int *maxdist*)

The local analog to fold().

Computes the minimum free energy structure including only base pairs with a span smaller than 'maxdist'

Parameters:

string

structure

maxdist

6.19.1.2 float Lfoldz (const char ∗ *string*, char ∗ *structure*, int *maxdist*, int *zsc*, double *min_z*)

Parameters:

string

structure

maxdist

zsc

min_z

6.20 Partition functions for locally stable secondary structures

Collaboration diagram for Partition functions for locally stable secondary structures:

Functions

- void update_pf_paramsLP (int length)
- plist ∗ pfl_fold (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, struct plist ∗∗dpp2, FILE ∗pUfp, FILE ∗spup)

  Compute partition functions for locally stable secondary structures.
- plist ∗ pfl_fold_par (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, struct plist ∗∗dpp2, FILE ∗pUfp, FILE ∗spup, pf_paramT ∗parameters)

  Compute partition functions for locally stable secondary structures.
- void putoutpU_prob (double ∗∗pU, int length, int ulength, FILE ∗fp, int energies)

  Writes the unpaired probabilities (pU) or opening energies into a file.
- void putoutpU_prob_bin (double ∗∗pU, int length, int ulength, FILE ∗fp, int energies)

  Writes the unpaired probabilities (pU) or opening energies into a binary file.

6.20.1 Function Documentation

6.20.1.1 void update_pf_paramsLP (int *length*)

Parameters:

length

6.20.1.2 plist∗ pfl_fold (char ∗ *sequence*, int *winSize*, int *pairSize*, float *cutoffb*, double ∗∗ *pU*, struct plist ∗∗ *dpp2*, FILE ∗ *pUfp*, FILE ∗ *spup*)

Compute partition functions for locally stable secondary structures.

pfl_fold computes partition functions for every window of size 'winSize' possible in a RNA molecule, allowing only pairs with a span smaller than 'pairSize'. It returns the mean pair probabilities averaged over all windows containing the pair in 'pl'. 'winSize' should always be >= 'pairSize'. Note that in contrast to Lfold(), bases outside of the window do not influence the structure at all. Only probabilities higher than 'cutoffb' are kept.

If 'pU' is supplied (i.e is not the NULL pointer), pfl_fold() will also compute the mean probability that regions of length 'u' and smaller are unpaired. The parameter 'u' is supplied in 'pup[0][0]'. On return the 'pup' array will contain these probabilities, with the entry on 'pup[x][y]' containing the mean probability that x and the y-1 preceding bases are unpaired. The 'pU' array needs to be large enough to hold n+1 float∗ entries, where n is the sequence length.

If an array dpp2 is supplied, the probability of base pair (i,j) given that there already exists a base pair (i+1,j-1) is also computed and saved in this array. If pUfp is given (i.e. not NULL), pU is not saved but put out imediately. If spup is given (i.e. is not NULL), the pair probabilities in pl are not saved but put out imediately.

Parameters:

sequence RNA sequence

winSize size of the window

pairSize maximum size of base pair

cutoffb cutoffb for base pairs

pU array holding all unpaired probabilities

dpp2 array of dependent pair probabilities

pUfp file pointer for pU

spup file pointer for pair probabilities

Returns:

list of pair probabilities

6.20.1.3 void putoutpU_prob (double ∗∗ *pU*, int *length*, int *ulength*, FILE ∗ *fp*, int *energies*)

Writes the unpaired probabilities (pU) or opening energies into a file.

Can write either the unpaired probabilities (accessibilities) pU or the opening energies -log(pU)kT into a file

Parameters:

pU pair probabilities

length length of RNA sequence

ulength maximum length of unpaired stretch

fp file pointer of destination file

energies switch to put out as opening energies

6.20.1.4 void putoutpU_prob_bin (double ∗∗ *pU*, int *length*, int *ulength*, FILE ∗ *fp*, int *energies*)

Writes the unpaired probabilities (pU) or opening energies into a binary file.

Can write either the unpaired probabilities (accessibilities) pU or the opening energies -log(pU)kT into a file

Parameters:

pU	pair probabilities
length	length of RNA sequence
ulength	maximum length of unpaired stretch
fp	file pointer of destination file
energies	switch to put out as opening energies

6.21 Local MFE consensus structures for Sequence Alignments

Collaboration diagram for Local MFE consensus structures for Sequence Alignments:

Functions

- float aliLfold (const char ∗∗strings, char ∗structure, int maxdist)

6.21.1 Function Documentation

6.21.1.1 float aliLfold (const char ∗∗ *strings*, char ∗ *structure*, int *maxdist*)

Parameters:

strings

structure

maxdist

Returns:

6.22 Change and Precalculate Energy Parameter Sets and Boltzmann Factors

Collaboration diagram for Change and Precalculate Energy Parameter Sets and Boltzmann Factors:

Files

- file params.h

Modules

- Reading/Writing energy parameter sets from/to File

Functions

- paramT ∗ scale_parameters (void)

  Get precomputed energy contributions for all the known loop types.

- paramT ∗ get_scaled_parameters (double temperature, model_detailsT md)

  Get precomputed energy contributions for all the known loop types.

- pf_paramT ∗ get_scaled_pf_parameters (void)
- pf_paramT ∗ get_boltzmann_factors (double temperature, double betaScale, model_detailsT md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.

- pf_paramT ∗ get_boltzmann_factor_copy (pf_paramT ∗parameters)

  Get a copy of already precomputed Boltzmann factors.

- pf_paramT ∗ get_scaled_alipf_parameters (unsigned int n_seq)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant).

- PUBLIC pf_paramT ∗ get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, model_detailsT md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.

6.22.1 Function Documentation

6.22.1.1 paramT∗ scale_parameters (void)

Get precomputed energy contributions for all the known loop types.

Note:

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See get_scaled_parameters() for a completely threadsafe implementation.

Returns:

A set of precomputed energy contributions

6.22.1.2 paramT∗ get_scaled_parameters (double *temperature*, model_detailsT *md*)

Get precomputed energy contributions for all the known loop types.

Call this function to retrieve precomputed energy contributions, i.e. scaled according to the temperature passed. Furthermore, this function assumes a data structure that contains the model details as well, such that subsequent folding recursions are able to retrieve the correct model settings

See also:

model_detailsT, set_model_details()

Parameters:

temperature The temperature in degrees Celcius

md The model details

Returns:

precomputed energy contributions and model settings

6.22.1.3 pf_paramT∗ get_scaled_pf_parameters (void)

get a datastructure of type pf_paramT which contains the Boltzmann weights of several energy parameters scaled according to the current temperature

Returns:

The datastructure containing Boltzmann weights for use in partition function calculations

6.22.1.4 pf_paramT$*$ get_boltzmann_factors (double *temperature*, double *betaScale*, model_detailsT *md*, double *pf_scale*)

Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.

This function returns a data structure that contains all necessary precalculated Boltzmann factors for each loop type contribution.

In contrast to get_scaled_pf_parameters(), this function enables setting of independent temperatures for both, the individual energy contributions as well as the thermodynamic temperature used in $exp(-\Delta G/kT)$

See also:

get_scaled_pf_parameters(), get_boltzmann_factor_copy()

Parameters:

temperature The temperature in degrees Celcius used for (re-)scaling the energy contributions

betaScale A scaling value that is used as a multiplication factor for the absolute temperature of the system

md The model details to be used

pf_scale The scaling factor for the Boltzmann factors

Returns:

A set of precomputed Boltzmann factors

6.22.1.5 pf_paramT$*$ get_boltzmann_factor_copy (pf_paramT $*$ *parameters*)

Get a copy of already precomputed Boltzmann factors.

See also:

get_boltzmann_factors(), get_scaled_pf_parameters()

Parameters:

parameters The input data structure that shall be copied

Returns:

A copy of the provided Boltzmann factor dataset

6.23 Reading/Writing energy parameter sets from/to File

Collaboration diagram for Reading/Writing energy parameter sets from/to File:

Files

- file read_epars.h

Modules

- Converting energy parameter files

Functions

- void read_parameter_file (const char fname[ ])

  Read energy parameters from a file.

- void write_parameter_file (const char fname[ ])

  Write energy parameters to a file.

6.23.1 Function Documentation

6.23.1.1 void read_parameter_file (const char *fname*[ ])

Read energy parameters from a file.

Parameters:

fname The path to the file containing the energy parameters

6.23.1.2 void write_parameter_file (const char *fname*[ ])

Write energy parameters to a file.

Parameters:

fname A filename (path) for the file where the current energy parameters will be written to

6.24 Converting energy parameter files

Collaboration diagram for Converting energy parameter files:

Files

- file convert_epars.h

    Functions and definitions for energy parameter file format conversion.

Defines

- #define VRNA_CONVERT_OUTPUT_ALL 1U
- #define VRNA_CONVERT_OUTPUT_HP 2U
- #define VRNA_CONVERT_OUTPUT_STACK 4U
- #define VRNA_CONVERT_OUTPUT_MM_HP 8U
- #define VRNA_CONVERT_OUTPUT_MM_INT 16U
- #define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U
- #define VRNA_CONVERT_OUTPUT_MM_INT_23 64U
- #define VRNA_CONVERT_OUTPUT_MM_MULTI 128U
- #define VRNA_CONVERT_OUTPUT_MM_EXT 256U
- #define VRNA_CONVERT_OUTPUT_DANGLE5 512U
- #define VRNA_CONVERT_OUTPUT_DANGLE3 1024U
- #define VRNA_CONVERT_OUTPUT_INT_11 2048U
- #define VRNA_CONVERT_OUTPUT_INT_21 4096U
- #define VRNA_CONVERT_OUTPUT_INT_22 8192U
- #define VRNA_CONVERT_OUTPUT_BULGE 16384U
- #define VRNA_CONVERT_OUTPUT_INT 32768U
- #define VRNA_CONVERT_OUTPUT_ML 65536U
- #define VRNA_CONVERT_OUTPUT_MISC 131072U
- #define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U
- #define VRNA_CONVERT_OUTPUT_VANILLA 524288U
- #define VRNA_CONVERT_OUTPUT_NINIO 1048576U
- #define VRNA_CONVERT_OUTPUT_DUMP 2097152U

Functions

- void convert_parameter_file (const char ∗iname, const char ∗oname, unsigned int options)

6.24.1 Define Documentation

6.24.1.1 #define VRNA_CONVERT_OUTPUT_ALL 1U

Flag to indicate printing of a complete parameter set

6.24.1.2 #define VRNA_CONVERT_OUTPUT_HP 2U

Flag to indicate printing of hairpin contributions

6.24.1.3 #define VRNA_CONVERT_OUTPUT_STACK 4U

Flag to indicate printing of base pair stack contributions

6.24.1.4 #define VRNA_CONVERT_OUTPUT_MM_HP 8U

Flag to indicate printing of hairpin mismatch contribution

6.24.1.5 #define VRNA_CONVERT_OUTPUT_MM_INT 16U

Flag to indicate printing of interior loop mismatch contribution

6.24.1.6 #define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U

Flag to indicate printing of 1:n interior loop mismatch contribution

6.24.1.7 #define VRNA_CONVERT_OUTPUT_MM_INT_23 64U

Flag to indicate printing of 2:3 interior loop mismatch contribution

6.24.1.8 #define VRNA_CONVERT_OUTPUT_MM_MULTI 128U

Flag to indicate printing of multi loop mismatch contribution

6.24.1.9 #define VRNA_CONVERT_OUTPUT_MM_EXT 256U

Flag to indicate printing of exterior loop mismatch contribution

6.24.1.10 #define VRNA_CONVERT_OUTPUT_DANGLE5 512U

Flag to indicate printing of 5' dangle conctribution

6.24.1.11 #define VRNA_CONVERT_OUTPUT_DANGLE3 1024U

Flag to indicate printing of 3' dangle contribution

6.24.1.12 #define VRNA_CONVERT_OUTPUT_INT_11 2048U

Flag to indicate printing of 1:1 interior loop contribution

6.24.1.13 #define VRNA_CONVERT_OUTPUT_INT_21 4096U

Flag to indicate printing of 2:1 interior loop contribution

6.24.1.14 #define VRNA_CONVERT_OUTPUT_INT_22 8192U

Flag to indicate printing of 2:2 interior loop contribution

6.24.1.15 #define VRNA_CONVERT_OUTPUT_BULGE 16384U

Flag to indicate printing of bulge loop contribution

6.24.1.16 #define VRNA_CONVERT_OUTPUT_INT 32768U

Flag to indicate printing of interior loop contribution

6.24.1.17 #define VRNA_CONVERT_OUTPUT_ML 65536U

Flag to indicate printing of multi loop contribution

6.24.1.18 #define VRNA_CONVERT_OUTPUT_MISC 131072U

Flag to indicate printing of misc contributions (such as terminalAU)

6.24.1.19 #define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U

Flag to indicate printing of special hairpin contributions (tri-, tetra-, hexa-loops)

6.24.1.20 #define VRNA_CONVERT_OUTPUT_VANILLA 524288U

Flag to indicate printing of given parameters only

Note:

This option overrides all other output options, except VRNA_CONVERT_-OUTPUT_DUMP !

6.24.1.21 #define VRNA_CONVERT_OUTPUT_NINIO 1048576U

Flag to indicate printing of interior loop asymmetry contribution

6.24.1.22 #define VRNA_CONVERT_OUTPUT_DUMP 2097152U

Flag to indicate dumping the energy contributions from the library instead of an input file

6.24.2 Function Documentation

6.24.2.1 void convert_parameter_file (const char $*$ *iname*, const char $*$ *oname*, unsigned int *options*)

Convert/dump a Vienna 1.8.4 formatted energy parameter file

The options argument allows to control the different output modes.

Currently available options are:

VRNA_CONVERT_OUTPUT_ALL, VRNA_CONVERT_OUTPUT_HP, VRNA_-CONVERT_OUTPUT_STACK

VRNA_CONVERT_OUTPUT_MM_HP, VRNA_CONVERT_OUTPUT_MM_INT, VRNA_CONVERT_OUTPUT_MM_INT_1N

VRNA_CONVERT_OUTPUT_MM_INT_23, VRNA_CONVERT_OUTPUT_MM_-MULTI, VRNA_CONVERT_OUTPUT_MM_EXT

VRNA_CONVERT_OUTPUT_DANGLE5, VRNA_CONVERT_OUTPUT_-DANGLE3, VRNA_CONVERT_OUTPUT_INT_11

VRNA_CONVERT_OUTPUT_INT_21, VRNA_CONVERT_OUTPUT_INT_22, VRNA_CONVERT_OUTPUT_BULGE

VRNA_CONVERT_OUTPUT_INT, VRNA_CONVERT_OUTPUT_ML, VRNA_-CONVERT_OUTPUT_MISC

VRNA_CONVERT_OUTPUT_SPECIAL_HP, VRNA_CONVERT_OUTPUT_-VANILLA, VRNA_CONVERT_OUTPUT_NINIO

VRNA_CONVERT_OUTPUT_DUMP

The defined options are fine for bitwise compare- and assignment-operations, e. g.: pass a collection of options as a single value like this:

```
convert_parameter_file(ifile, ofile, option_1 | option_2 | option_n)
```

Parameters:

iname The input file name (If NULL input is read from stdin)

oname The output file name (If NULL output is written to stdout)

options The options (as described above)

6.25 Energy evaluation

Collaboration diagram for Energy evaluation:This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

Functions

- float energy_of_structure (const char ∗string, const char ∗structure, int verbosity_level)

    Calculate the free energy of an already folded RNA using global model detail settings.

- float energy_of_struct_par (const char ∗string, const char ∗structure, paramT ∗parameters, int verbosity_level)

    Calculate the free energy of an already folded RNA.

- float energy_of_circ_structure (const char ∗string, const char ∗structure, int verbosity_level)

    Calculate the free energy of an already folded circular RNA.

- float energy_of_circ_struct_par (const char ∗string, const char ∗structure, paramT ∗parameters, int verbosity_level)

    Calculate the free energy of an already folded circular RNA.

- int energy_of_structure_pt (const char ∗string, short ∗ptable, short ∗s, short ∗s1, int verbosity_level)

    Calculate the free energy of an already folded RNA.

- int energy_of_struct_pt_par (const char ∗string, short ∗ptable, short ∗s, short ∗s1, paramT ∗parameters, int verbosity_level)

    Calculate the free energy of an already folded RNA.

Variables

- int eos_debug

    verbose info from energy_of_struct

6.25.1 Detailed Description

This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

6.25.2 Function Documentation

6.25.2.1 float energy_of_structure (const char $*$ *string*, const char $*$ *structure*, int *verbosity_level*)

Calculate the free energy of an already folded RNA using global model detail settings.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Note:

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See energy_of_struct_par() for a completely threadsafe implementation.

See also:

energy_of_struct_par(), energy_of_circ_structure()

Parameters:

string RNA sequence

structure secondary structure in dot-bracket notation

verbosity_level a flag to turn verbose output on/off

Returns:

the free energy of the input structure given the input sequence in kcal/mol

6.25.2.2 float energy_of_struct_par (const char $*$ *string*, const char $*$ *structure*, paramT $*$ *parameters*, int *verbosity_level*)

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

See also:

energy_of_circ_structure(), energy_of_structure_pt(), get_scaled_parameters()

Parameters:

string RNA sequence in uppercase letters

structure Secondary structure in dot-bracket notation

parameters A data structure containing the prescaled energy contributions and the model details.

verbosity_level A flag to turn verbose output on/off

Returns:

The free energy of the input structure given the input sequence in kcal/mol

6.25.2.3 float energy_of_circ_structure (const char ∗ *string*, const char ∗ *structure*, int *verbosity_level*)

Calculate the free energy of an already folded circular RNA.

Note:

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See energy_of_circ_struct_par() for a completely threadsafe implementation.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

See also:

energy_of_circ_struct_par(), energy_of_struct_par()

Parameters:

string RNA sequence

structure Secondary structure in dot-bracket notation

verbosity_level A flag to turn verbose output on/off

Returns:

The free energy of the input structure given the input sequence in kcal/mol

6.25.2.4 float energy_of_circ_struct_par (const char ∗ *string*, const char ∗ *structure*, paramT ∗ *parameters*, int *verbosity_level*)

Calculate the free energy of an already folded circular RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

See also:

energy_of_struct_par(), get_scaled_parameters()

Parameters:

string RNA sequence

structure Secondary structure in dot-bracket notation

parameters A data structure containing the prescaled energy contributions and the model details.

verbosity_level A flag to turn verbose output on/off

Returns:

The free energy of the input structure given the input sequence in kcal/mol

6.25.2.5 int energy_of_structure_pt (const char $*$ *string*, short $*$ *ptable*, short $*$ *s*, short $*$ *s1*, int *verbosity_level*)

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

Note:

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See energy_of_struct_pt_par() for a completely threadsafe implementation.

See also:

make_pair_table(), energy_of_struct_pt_par()

Parameters:

string RNA sequence

ptable the pair table of the secondary structure

s encoded RNA sequence

s1 encoded RNA sequence

verbosity_level a flag to turn verbose output on/off

Returns:

the free energy of the input structure given the input sequence in 10kcal/mol

6.25.2.6 int energy_of_struct_pt_par (const char $*$ *string*, short $*$ *ptable*, short $*$ *s*, short $*$ *s1*, paramT $*$ *parameters*, int *verbosity_level*)

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

See also:

make_pair_table(), energy_of_struct_par(), get_scaled_parameters()

Parameters:

string RNA sequence in uppercase letters

ptable The pair table of the secondary structure

s Encoded RNA sequence

s1 Encoded RNA sequence

parameters A data structure containing the prescaled energy contributions and the model details.

verbosity_level A flag to turn verbose output on/off

Returns:

The free energy of the input structure given the input sequence in 10kcal/mol

6.26 Searching Sequences for Predefined Structures

Collaboration diagram for Searching Sequences for Predefined Structures:

Files

- file inverse.h

  Inverse folding routines.

Functions

- float inverse_fold (char ∗start, const char ∗target)

  Find sequences with predefined structure.

- float inverse_pf_fold (char ∗start, const char ∗target)

  Find sequence that maximizes probability of a predefined structure.

Variables

- char ∗ symbolset

  This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.

- float final_cost
- int give_up
- int inv_verbose

6.26.1 Function Documentation

6.26.1.1 float inverse_fold (char ∗ *start*, const char ∗ *target*)

Find sequences with predefined structure.

This function searches for a sequence with minimum free energy structure provided in the parameter 'target', starting with sequence 'start'. It returns 0 if the search was successful, otherwise a structure distance in terms of the energy difference between the search result and the actual target 'target' is returned. The found sequence is returned in 'start'. If give_up is set to 1, the function will return as soon as it is clear that the search will be unsuccessful, this speeds up the algorithm if you are only interested in exact solutions.

Parameters:

start The start sequence

target The target secondary structure in dot-bracket notation

Returns:

The distance to the target in case a search was unsuccessful, 0 otherwise

6.26.1.2 float inverse_pf_fold (char $*$ *start*, const char $*$ *target*)

Find sequence that maximizes probability of a predefined structure.

This function searches for a sequence with maximum probability to fold into the provided structure 'target' using the partition function algorithm. It returns $-kT \cdot \log(p)$ where p is the frequency of 'target' in the ensemble of possible structures. This is usually much slower than inverse_fold().

Parameters:

start The start sequence

target The target secondary structure in dot-bracket notation

Returns:

The distance to the target in case a search was unsuccessful, 0 otherwise

6.26.2 Variable Documentation

6.26.2.1 float final_cost

when to stop inverse_pf_fold()

6.26.2.2 int give_up

default 0: try to minimize structure distance even if no exact solution can be found

6.26.2.3 int inv_verbose

print out substructure on which inverse_fold() fails

6.27 Classified Dynamic Programming

Collaboration diagram for Classified Dynamic Programming:

Modules

- Distance based partitioning of the Secondary Structure Space

  Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

- Compute the Density of States

6.28 Distance based partitioning of the Secondary Structure Space

Collaboration diagram for Distance based partitioning of the Secondary Structure Space:Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

Modules

- Calculating MFE representatives of a Distance Based Partitioning
- Calculate Partition Functions of a Distance Based Partitioning
- Stochastic Backtracking of Structures from Distance Based Partitioning

    Contains functions related to stochastic backtracking from a specified distance class.

6.28.1 Detailed Description

Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

All functions related to this group implement the basic recursions for MFE folding, partition function computation and stochastic backtracking with a *classified dynamic programming* approach. The secondary structure space is divided into partitions according to the base pair distance to two given reference structures and all relevant properties are calculated for each of the resulting partitions

See also:

For further details have a look into lorenz:2009

6.29 Calculating MFE representatives of a Distance Based Partitioning

Collaboration diagram for Calculating MFE representatives of a Distance Based Partitioning:

Files

- file 2Dfold.h

Functions

- TwoDfold_vars ∗ get_TwoDfold_variables (const char ∗seq, const char ∗structure1, const char ∗structure2, int circ)

  Get a structure of type TwoDfold_vars prefilled with current global settings.

- void destroy_TwoDfold_variables (TwoDfold_vars ∗our_variables)

  Destroy a TwoDfold_vars datastructure without memory loss.

- TwoDfold_solution ∗ TwoDfoldList (TwoDfold_vars ∗vars, int distance1, int distance2)

  Compute MFE's and representative for distance partitioning.

- char ∗ TwoDfold_backtrack_f5 (unsigned int j, int k, int l, TwoDfold_vars ∗vars)

  Backtrack a minimum free energy structure from a 5' section of specified length.

6.29.1 Function Documentation

6.29.1.1 TwoDfold_vars∗ get_TwoDfold_variables (const char ∗ *seq*, const char ∗ *structure1*, const char ∗ *structure2*, int *circ*)

Get a structure of type TwoDfold_vars prefilled with current global settings.

This function returns a datastructure of type TwoDfold_vars. The data fields inside the TwoDfold_vars are prefilled by global settings and all memory allocations necessary to start a computation are already done for the convenience of the user

Note:

Make sure that the reference structures are compatible with the sequence according to Watson-Crick- and Wobble-base pairing

See also:

destroy_TwoDfold_variables(), TwoDfold(), TwoDfold_circ

Parameters:

seq The RNA sequence

structure1 The first reference structure in dot-bracket notation

structure2 The second reference structure in dot-bracket notation

circ A switch to indicate the assumption to fold a circular instead of linear RNA (0=OFF, 1=ON)

Returns:

A datastructure prefilled with folding options and allocated memory

6.29.1.2 void destroy_TwoDfold_variables (TwoDfold_vars ∗ *our_variables*)

Destroy a TwoDfold_vars datastructure without memory loss.

This function free's all allocated memory that depends on the datastructure given.

See also:

get_TwoDfold_variables()

Parameters:

our_variables A pointer to the datastructure to be destroyed

6.29.1.3 TwoDfold_solution∗ TwoDfoldList (TwoDfold_vars ∗ *vars*, int *distance1*, int *distance2*)

Compute MFE's and representative for distance partitioning.

This function computes the minimum free energies and a representative secondary structure for each distance class according to the two references specified in the datastructure 'vars'. The maximum basepair distance to each of both references may be set by the arguments 'distance1' and 'distance2', respectively. If both distance arguments are set to '-1', no restriction is assumed and the calculation is performed for each distance class possible.

The returned list contains an entry for each distance class. If a maximum basepair distance to either of the references was passed, an entry with k=l=-1 will be appended in the list, denoting the class where all structures exceeding the maximum will be thrown into. The end of the list is denoted by an attribute value of INF in the k-attribute of the list entry.

See also:

get_TwoDfold_variables(), destroy_TwoDfold_variables(), TwoDfold_solution

Parameters:

vars the datastructure containing all predefined folding attributes

distance1 maximum distance to reference1 (-1 means no restriction)

distance2 maximum distance to reference2 (-1 means no restriction)

6.29.1.4 char∗ TwoDfold_backtrack_f5 (unsigned int *j*, int *k*, int *l*, TwoDfold_vars ∗ *vars*)

Backtrack a minimum free energy structure from a 5' section of specified length.

This function allows to backtrack a secondary structure beginning at the 5' end, a specified length and residing in a specific distance class. If the argument 'k' gets a value of -1, the structure that is backtracked is assumed to reside in the distance class where all structures exceeding the maximum basepair distance specified in TwoDfoldList() belong to.

Note:

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to TwoDfoldList() preceding this function is mandatory!

See also:

TwoDfoldList(), get_TwoDfold_variables(), destroy_TwoDfold_variables()

Parameters:

j The length in nucleotides beginning from the 5' end

k distance to reference1 (may be -1)

l distance to reference2

vars the datastructure containing all predefined folding attributes

6.30 Calculate Partition Functions of a Distance Based Partitioning

Collaboration diagram for Calculate Partition Functions of a Distance Based Partitioning:

Files

- file 2Dpfold.h

Functions

- TwoDpfold_vars ∗ get_TwoDpfold_variables (const char ∗seq, const char ∗structure1, char ∗structure2, int circ)

  Get a datastructure containing all necessary attributes and global folding switches.

- TwoDpfold_vars ∗ get_TwoDpfold_variables_from_MFE (TwoDfold_vars ∗mfe_vars)

  Get the datastructure containing all necessary attributes and global folding switches from a pre-filled mfe-datastructure.

- void destroy_TwoDpfold_variables (TwoDpfold_vars ∗vars)

  Free all memory occupied by a TwoDpfold_vars datastructure.

- TwoDpfold_solution ∗ TwoDpfoldList (TwoDpfold_vars ∗vars, int maxDistance1, int maxDistance2)

  Compute the partition function for all distance classes.

6.30.1 Function Documentation

6.30.1.1 TwoDpfold_vars∗ get_TwoDpfold_variables (const char ∗ *seq*, const char ∗ *structure1*, char ∗ *structure2*, int *circ*)

Get a datastructure containing all necessary attributes and global folding switches.

This function prepares all necessary attributes and matrices etc which are needed for a call of TwoDpfoldList. A snapshot of all current global model switches (dangles, temperature and so on) is done and stored in the returned datastructure. Additionally, all matrices that will hold the partition function values are prepared.

Parameters:

seq the RNA sequence in uppercase format with letters from the alphabet {AUCG}

structure1 the first reference structure in dot-bracket notation

structure2 the second reference structure in dot-bracket notation

circ a switch indicating if the sequence is linear (0) or circular (1)

Returns:

the datastructure containing all necessary partition function attributes

6.30.1.2 TwoDpfold_vars∗ get_TwoDpfold_variables_from_MFE (TwoDfold_vars ∗ *mfe_vars*)

Get the datastructure containing all necessary attributes and global folding switches from a pre-filled mfe-datastructure.

This function actually does the same as get_TwoDpfold_variables but takes its switches and settings from a pre-filled MFE equivalent datastructure

See also:

get_TwoDfold_variables(), get_TwoDpfold_variables()

Parameters:

mfe_vars the pre-filled mfe datastructure

Returns:

the datastructure containing all necessary partition function attributes

6.30.1.3 void destroy_TwoDpfold_variables (TwoDpfold_vars ∗ *vars*)

Free all memory occupied by a TwoDpfold_vars datastructure.

This function free's all memory occupied by a datastructure obtained from from get_-TwoDpfold_variables() or get_TwoDpfold_variables_from_MFE()

See also:

get_TwoDpfold_variables(), get_TwoDpfold_variables_from_MFE()

Parameters:

vars the datastructure to be free'd

6.30.1.4 TwoDpfold_solution∗ TwoDpfoldList (TwoDpfold_vars ∗ *vars*, int *maxDistance1*, int *maxDistance2*)

Compute the partition function for all distance classes.

This function computes the partition functions for all distance classes according the two reference structures specified in the datastructure 'vars'. Similar to TwoDfoldList() the arguments maxDistance1 and maxDistance2 specify the maximum distance to both reference structures. A value of '-1' in either of them makes the appropriate distance restrictionless, i.e. all basepair distancies to the reference are taken into account during computation. In case there is a restriction, the returned solution contains an entry where the attribute k=l=-1 contains the partition function for all structures exceeding the restriction. A values of INF in the attribute 'k' of the returned list denotes the end of the list

See also:

get_TwoDpfold_variables(), destroy_TwoDpfold_variables(), TwoDpfold_-solution

Parameters:

vars	the datastructure containing all necessary folding attributes and matrices
maxDistance1	the maximum basepair distance to reference1 (may be -1)
maxDistance2	the maximum basepair distance to reference2 (may be -1)

Returns:

a list of partition funtions for the appropriate distance classes

6.31 Stochastic Backtracking of Structures from Distance Based Partitioning

Collaboration diagram for Stochastic Backtracking of Structures from Distance Based Partitioning:Contains functions related to stochastic backtracking from a specified distance class.

Functions

- char ∗ TwoDpfold_pbacktrack (TwoDpfold_vars ∗vars, int d1, int d2)

  Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

- char ∗ TwoDpfold_pbacktrack5 (TwoDpfold_vars ∗vars, int d1, int d2, unsigned int length)

  Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

6.31.1 Detailed Description

Contains functions related to stochastic backtracking from a specified distance class.

6.31.2 Function Documentation

6.31.2.1 char∗ TwoDpfold_pbacktrack (TwoDpfold_vars ∗ *vars*, int *d1*, int *d2*)

Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

If the argument ’d1’ is set to ’-1’, the structure will be backtracked in the distance class where all structures exceeding the maximum basepair distance to either of the references reside.

Precondition:

The argument ’vars’ must contain precalculated partition function matrices, i.e. a call to TwoDpfoldList() preceding this function is mandatory!

See also:

TwoDpfoldList()

Parameters:

← ***vars*** the datastructure containing all necessary folding attributes and matrices

← ***d1*** the distance to reference1 (may be -1)

← ***d2*** the distance to reference2

Returns:

A sampled secondary structure in dot-bracket notation

6.31.2.2 char∗ TwoDpfold_pbacktrack5 (TwoDpfold_vars ∗ *vars*, int *d1*, int *d2*, unsigned int *length*)

Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

This function does essentially the same as TwoDpfold_pbacktrack with the only difference that partial structures, i.e. structures beginning from the 5' end with a specified length of the sequence, are backtracked

Note:

This function does not work (since it makes no sense) for circular RNA sequences!

Precondition:

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to TwoDpfoldList() preceding this function is mandatory!

See also:

TwoDpfold_pbacktrack(), TwoDpfoldList()

Parameters:

← ***vars*** the datastructure containing all necessary folding attributes and matrices

← ***d1*** the distance to reference1 (may be -1)

← ***d2*** the distance to reference2

← ***length*** the length of the structure beginning from the 5' end

Returns:

A sampled secondary structure in dot-bracket notation

6.32 Compute the Density of States

Collaboration diagram for Compute the Density of States:

Variables

- int density_of_states [MAXDOS+1]

  The Density of States.

6.32.1 Variable Documentation

6.32.1.1 int density_of_states[MAXDOS+1]

The Density of States.

This array contains the density of states for an RNA sequences after a call to subopt_-par(), subopt() or subopt_circ().

Precondition:

Call one of the functions subopt_par(), subopt() or subopt_circ() prior accessing the contents of this array

See also:

subopt_par(), subopt(), subopt_circ()

6.33 Parsing and Comparing - Functions to Manipulate Structures

6.34 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/2Dfold.h

Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.

6.35 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/2Dpfold.h

Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.

6.36 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/alifold.h

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments Consensus structures can be predicted by a modified version of the fold() algorithm that takes a set of aligned sequences instead of a single sequence. The energy function consists of the mean energy averaged over the sequences, plus a covariance term that favors pairs with consistent and compensatory mutations and penalizes pairs that cannot be formed by all structures. For details see hofacker:2002 and bernhart:2008.

6.37 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/cofold.h

6.38 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/convert_epars.h

Convert energy parameter files into the latest format. To preserve some backward compatibility the RNAlib also provides functions to convert energy parameter files from the format used in version 1.4-1.8 into the new format used since version 2.0

6.39 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/fold.h

Collaboration diagram for /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/fold.h:This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures. The library provides a fast dynamic programming minimum free energy folding algorithm as described in zuker:1981. All relevant parts that directly implement the "Zuker & Stiegler" algorithm for single sequences are described in this section.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See hofacker:2006 for further details.

Nevertheless, the RNAlib also provides interfaces for the prediction of consensus MFE structures of sequence alignments, MFE structure for two hybridized sequences, local optimal structures and many more. For those more specialized variants of MFE folding routines, please consult the appropriate subsections (Modules) as listed above.

6.40 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/inverse.h

We provide two functions that search for sequences with a given structure, thereby inverting the folding routines.

6.41 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/Lfold.h

Local structures can be predicted by a modified version of the fold() algorithm that restricts the span of all base pairs.

6.42 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/LPfold.h

6.43 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/params.h

All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s). This module covers all relevant functions for precalculation of the energy parameters necessary for the folding routines provided by RNAlib. Furthermore, the energy parameter set in the RNAlib can be easily exchanged by a user-defined one. It is also possible to write the current energy parameter set into a text file.

6.44 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/part_func.h

This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities. Instead of the minimum free energy structure the partition function of all possible structures and from that the pairing probability for every possible pair can be calculated, using a dynamic programming algorithm as described in mccaskill:1990.

6.45 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/part_func_co.h

Partition Function Cofolding. To simplify the implementation the partition function computation is done internally in a null model that does not include the duplex initiation energy, i.e. the entropic penalty for producing a dimer from two monomers). The resulting free energies and pair probabilities are initially relative to that null model. In a second step the free energies can be corrected to include the dimerization penalty, and the pair probabilities can be divided into the conditional pair probabilities given that a re dimer is formed or not formed. See bernhart:2006 for further details.

6.46 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/part_func_up.h

Partition Function Cofolding as a stepwise process.

6.47 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/read_epars.h

Read and Write energy parameter sets from and to text files. A default set of parameters, identical to the one described in mathews:2004 and turner:2010, is compiled into the library.

6.48 Enumerating Suboptimal Structures

Collaboration diagram for Enumerating Suboptimal Structures:

Chapter 7

RNAlib-2.1.2 Data Structure Documentation

7.1 bondT Struct Reference

Base pair.

7.1.1 Detailed Description

Base pair.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.2 bondTEn Struct Reference

Base pair with associated energy.

7.2.1 Detailed Description

Base pair with associated energy.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.3 constrain Struct Reference

constraints for cofolding

7.3.1 Detailed Description

constraints for cofolding

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.4 COORDINATE Struct Reference

this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE

7.4.1 Detailed Description

this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.5 cpair Struct Reference

this datastructure is used as input parameter in functions of PS_dot.c

7.5.1 Detailed Description

this datastructure is used as input parameter in functions of PS_dot.c

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.6 INTERVAL Struct Reference

Sequence interval stack element used in subopt.c.

7.6.1 Detailed Description

Sequence interval stack element used in subopt.c.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.7 model_detailsT Struct Reference

The data structure that contains the complete model details used throughout the calculations.

Data Fields

- int dangles

  Specifies the dangle model used in any energy evaluation (0,1,2 or 3).

- int special_hp

  Include special hairpin contributions for tri, tetra and hexaloops.

- int noLP

  Only consider canonical structures, i.e. no 'lonely' base pairs.

- int noGU

  Do not allow GU pairs.

- int noGUclosure

  Do not allow loops to be closed by GU pair.

- int logML

  Use logarithmic scaling for multi loops.

- int circ

  Assume molecule to be circular.

- int gquad

  Include G-quadruplexes in structure prediction.

7.7.1 Detailed Description

The data structure that contains the complete model details used throughout the calculations.

7.7.2 Field Documentation

7.7.2.1 int model_detailsT::dangles

Specifies the dangle model used in any energy evaluation (0,1,2 or 3).

Note:

Some function do not implement all dangle model but only a subset of (0,1,2,3). Read the documentaion of the particular recurrences or energy evaluation function for information about the provided dangle model.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.8 PAIR Struct Reference

Base pair data structure used in subopt.c.

7.8.1 Detailed Description

Base pair data structure used in subopt.c.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.9 pair_info Struct Reference

A base pair info structure.

Data Fields

- unsigned i

  nucleotide position i

- unsigned j

  nucleotide position j

- float p

  Probability.

- float ent

  Pseudo entropy for $p(i, j) = S_i + S_j - p_ij * ln(p_ij)$.

- short bp [8]

  Frequencies of pair_types.

- char comp

  1 iff pair is in mfe structure

7.9.1 Detailed Description

A base pair info structure.

For each base pair (i,j) with i,j in [0, n-1] the structure lists:

- its probability 'p'
- an entropy-like measure for its well-definedness 'ent'
- the frequency of each type of pair in 'bp[]' + 'bp[0]' contains the number of non-compatible sequences + 'bp[1]' the number of CG pairs, etc.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.10 paramT Struct Reference

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for paramT:

Data Fields

- double temperature

  Temperature used for loop contribution scaling.

- model_detailsT model_details

  Model details to be used in the recursions.

7.10.1 Detailed Description

The datastructure that contains temperature scaled energy parameters.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.11 pf_paramT Struct Reference

The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.

Collaboration diagram for pf_paramT:

Data Fields

- double pf_scale

    Scaling factor to avoid over-/underflows.

- double temperature

    Temperature used for loop contribution scaling.

- double alpha

    Scaling factor for the thermodynamic temperature.

- model_detailsT model_details

    Model details to be used in the recursions.

7.11.1 Detailed Description

The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.

7.11.2 Field Documentation

7.11.2.1 double pf_paramT::alpha

Scaling factor for the thermodynamic temperature.

This allows for temperature scaling in Boltzmann factors independently from the energy contributions. The resulting Boltzmann factors are then computed by $e^{-E/(\alpha \cdot K \cdot T)}$

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.12 plist Struct Reference

this datastructure is used as input parameter in functions of PS_dot.h and others

7.12.1 Detailed Description

this datastructure is used as input parameter in functions of PS_dot.h and others

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.13 pu_contrib Struct Reference

contributions to p_u

Data Fields

- double ∗∗ H

  hairpin loops

- double ∗∗ I

  interior loops

- double ∗∗ M

  multi loops

- double ∗∗ E

  exterior loop

- int length

  length of the input sequence

- int w

  longest unpaired region

7.13.1 Detailed Description

contributions to p_u

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.14 pu_out Struct Reference

Collection of all free_energy of beeing unpaired values for output.

Data Fields

- int len

  sequence length

- int u_vals

  number of different -u values

- int contribs

  [-c "SHIME"]

- char $**$ header

  header line

- double $**$ u_values

  (the -u values $*$ [-c "SHIME"]) $*$ seq len

7.14.1 Detailed Description

Collection of all free_energy of beeing unpaired values for output.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.15 sect Struct Reference

Stack of partial structures for backtracking.

7.15.1 Detailed Description

Stack of partial structures for backtracking.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.16 SOLUTION Struct Reference

Solution element from subopt.c.

Data Fields

- float energy

  Free Energy of structure in kcal/mol.

- char ∗ structure

  Structure in dot-bracket notation.

7.16.1 Detailed Description

Solution element from subopt.c.

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.17 TwoDfold_solution Struct Reference

Solution element returned from TwoDfoldList.

Data Fields

- int k

  Distance to first reference.

- int l

  Distance to second reference.

- float en

  Free energy in kcal/mol.

- char ∗ s

  MFE representative structure in dot-bracket notation.

7.17.1 Detailed Description

Solution element returned from TwoDfoldList.

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'en' of type float representing the free energy in kcal/mol and an attribute 's' of type char∗ containg the secondary structure representative,

A value of INF in k denotes the end of a list

See also:

TwoDfoldList()

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.18 TwoDfold_vars Struct Reference

Variables compound for 2Dfold MFE folding.

Collaboration diagram for TwoDfold_vars:

Data Fields

- paramT ∗ P

  Precomputed energy parameters and model details.
- int do_backtrack

  Flag whether to do backtracing of the structure(s) or not.
- char ∗ ptype

  Precomputed array of pair types.
- char ∗ sequence

  The input sequence.
- short ∗ S1

  The input sequences in numeric form.
- unsigned int maxD1

  Maximum allowed base pair distance to first reference.
- unsigned int maxD2

  Maximum allowed base pair distance to second reference.
- unsigned int ∗ mm1

  Maximum matching matrix, reference struct 1 disallowed.
- unsigned int ∗ mm2

  Maximum matching matrix, reference struct 2 disallowed.
- int ∗ my_iindx

  Index for moving in quadratic distancy dimensions.
- unsigned int ∗ referenceBPs1

  Matrix containing number of basepairs of reference structure1 in interval [i,j].
- unsigned int ∗ referenceBPs2

  Matrix containing number of basepairs of reference structure2 in interval [i,j].
- unsigned int ∗ bpdist

  Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].

7.18.1 Detailed Description

Variables compound for 2Dfold MFE folding.

See also:

get_TwoDfold_variables(), destroy_TwoDfold_variables(), TwoDfoldList()

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.19 TwoDpfold_solution Struct Reference

Solution element returned from TwoDpfoldList.

Data Fields

- int k

  Distance to first reference.

- int l

  Distance to second reference.

- double q

  partition function

7.19.1 Detailed Description

Solution element returned from TwoDpfoldList.

This element contains the partition function for the appropriate kappa (k), lambda (l) neighborhood The datastructure contains two integer attributes ’k’ and ’l’ as well as an attribute ’q’ of type FLT_OR_DBL

A value of INF in k denotes the end of a list

See also:

TwoDpfoldList()

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

7.20 TwoDpfold_vars Struct Reference

Variables compound for 2Dfold partition function folding.

Collaboration diagram for TwoDpfold_vars:

Data Fields

- char ∗ ptype

  Precomputed array of pair types.

- char ∗ sequence

  The input sequence.

- short ∗ S1

  The input sequences in numeric form.

- unsigned int maxD1

  Maximum allowed base pair distance to first reference.

- unsigned int maxD2

  Maximum allowed base pair distance to second reference.

- int ∗ my_iindx

  Index for moving in quadratic distancy dimensions.

- int ∗ jindx

  Index for moving in the triangular matrix qm1.

- unsigned int ∗ referenceBPs1

  Matrix containing number of basepairs of reference structure1 in interval [i,j].

- unsigned int ∗ referenceBPs2

  Matrix containing number of basepairs of reference structure2 in interval [i,j].

- unsigned int ∗ bpdist

  Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].

- unsigned int ∗ mm1

  Maximum matching matrix, reference struct 1 disallowed.

- unsigned int ∗ mm2

  Maximum matching matrix, reference struct 2 disallowed.

7.20.1 Detailed Description

Variables compound for 2Dfold partition function folding.

See also:

get_TwoDpfold_variables(), get_TwoDpfold_variables_from_MFE(), destroy_-TwoDpfold_variables(), TwoDpfoldList()

The documentation for this struct was generated from the following file:

- /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h

Chapter 8

RNAlib-2.1.2 File Documentation

8.1 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/2Dfold.h File Reference

Include dependency graph for 2Dfold.h:

Functions

- TwoDfold_vars ∗ get_TwoDfold_variables (const char ∗seq, const char ∗structure1, const char ∗structure2, int circ)

  Get a structure of type TwoDfold_vars prefilled with current global settings.

- void destroy_TwoDfold_variables (TwoDfold_vars ∗our_variables)

  Destroy a TwoDfold_vars datastructure without memory loss.

- TwoDfold_solution ∗ TwoDfoldList (TwoDfold_vars ∗vars, int distance1, int distance2)

  Compute MFE's and representative for distance partitioning.

- char ∗ TwoDfold_backtrack_f5 (unsigned int j, int k, int l, TwoDfold_vars ∗vars)

  Backtrack a minimum free energy structure from a 5' section of specified length.

8.1.1 Detailed Description

8.2 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/2Dpfold.h File Reference

Include dependency graph for 2Dpfold.h:

Functions

- TwoDpfold_vars ∗ get_TwoDpfold_variables (const char ∗seq, const char ∗structure1, char ∗structure2, int circ)

  Get a datastructure containing all necessary attributes and global folding switches.

- TwoDpfold_vars ∗ get_TwoDpfold_variables_from_MFE (TwoDfold_vars ∗mfe_vars)

  Get the datastructure containing all necessary attributes and global folding switches from a pre-filled mfe-datastructure.

- void destroy_TwoDpfold_variables (TwoDpfold_vars ∗vars)

  Free all memory occupied by a TwoDpfold_vars datastructure.

- TwoDpfold_solution ∗ TwoDpfoldList (TwoDpfold_vars ∗vars, int maxDistance1, int maxDistance2)

  Compute the partition function for all distance classes.

- char ∗ TwoDpfold_pbacktrack (TwoDpfold_vars ∗vars, int d1, int d2)

  Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

- char ∗ TwoDpfold_pbacktrack5 (TwoDpfold_vars ∗vars, int d1, int d2, unsigned int length)

  Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

8.2.1 Detailed Description

8.3 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/alifold.h File Reference

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

Include dependency graph for alifold.h:

Functions

- void update_alifold_params (void)

  Update the energy parameters for alifold function.

- float alifold (const char ∗∗strings, char ∗structure)

  Compute MFE and according consensus structure of an alignment of sequences.

- float circalifold (const char ∗∗strings, char ∗structure)

  Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.

- void free_alifold_arrays (void)

  Free the memory occupied by MFE alifold functions.

- int get_mpi (char ∗Alseq[ ], int n_seq, int length, int ∗mini)

  Get the mean pairwise identity in steps from ?to?(ident).

- float ∗∗ readribosum (char ∗name)

  Read a ribosum or other user-defined scoring matrix.

- float energy_of_alistruct (const char ∗∗sequences, const char ∗structure, int n_-seq, float ∗energy)

  Calculate the free energy of a consensus structure given a set of aligned sequences.

- void encode_ali_sequence (const char ∗sequence, short ∗S, short ∗s5, short ∗s3, char ∗ss, unsigned short ∗as, int circ)

  Get arrays with encoded sequence of the alignment.

- void alloc_sequence_arrays (const char ∗∗sequences, short ∗∗∗S, short ∗∗∗S5, short ∗∗∗S3, unsigned short ∗∗∗a2s, char ∗∗∗Ss, int circ)

  Allocate memory for sequence array used to deal with aligned sequences.

- void free_sequence_arrays (unsigned int n_seq, short ∗∗∗S, short ∗∗∗S5, short ∗∗∗S3, unsigned short ∗∗∗a2s, char ∗∗∗Ss)

  Free the memory of the sequence arrays used to deal with aligned sequences.

- float alipf_fold_par (const char ∗∗sequences, char ∗structure, plist ∗∗pl, pf_-paramT ∗parameters, int calculate_bppm, int is_constrained, int is_circular)

- float alipf_fold (const char ∗∗sequences, char ∗structure, plist ∗∗pl)

  The partition function version of alifold() works in analogy to pf_fold(). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of pair_info structs. The list is terminated by the first entry with pi.i = 0.

- float alipf_circ_fold (const char ∗∗sequences, char ∗structure, plist ∗∗pl)
- double ∗ export_ali_bppm (void)

  Get a pointer to the base pair probability array.

- char ∗ alipbacktrack (double ∗prob)

  Sample a consensus secondary structure from the Boltzmann ensemble according its probability

  .

- int get_alipf_arrays (short ∗∗∗S_p, short ∗∗∗S5_p, short ∗∗∗S3_p, unsigned short ∗∗∗a2s_p, char ∗∗∗Ss_p, double ∗∗qb_p, double ∗∗qm_p, double ∗∗q1k_p, double ∗∗qln_p, short ∗∗pscore)

  Get pointers to (almost) all relavant arrays used in alifold's partition function computation.

Variables

- double cv_fact

  This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.

- double nc_fact

  This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

8.3.1 Detailed Description

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

8.3.2 Function Documentation

8.3.2.1 void update_alifold_params (void)

Update the energy parameters for alifold function.

Call this to recalculate the pair matrix and energy parameters after a change in folding parameters like temperature

8.4 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/cofold.h File Reference

MFE version of cofolding routines.

Include dependency graph for cofold.h:

Functions

- float cofold (const char ∗sequence, char ∗structure)

  Compute the minimum free energy of two interacting RNA molecules.

- float cofold_par (const char ∗string, char ∗structure, paramT ∗parameters, int is_constrained)

  Compute the minimum free energy of two interacting RNA molecules.

- void free_co_arrays (void)

  Free memory occupied by cofold().

- void update_cofold_params (void)

  Recalculate parameters.

- void export_cofold_arrays_gq (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗fc_p, int ∗∗ggg_p, int ∗∗indx_p, char ∗∗ptype_p)

  Export the arrays of partition function cofold (with gquadruplex support).

- void export_cofold_arrays (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗fc_p, int ∗∗indx_p, char ∗∗ptype_p)

  Export the arrays of partition function cofold.

- SOLUTION ∗ zukersubopt (const char ∗string)

  Compute Zuker type suboptimal structures.

- SOLUTION ∗ zukersubopt_par (const char ∗string, paramT ∗parameters)

  Compute Zuker type suboptimal structures.

- void get_monomere_mfes (float ∗e1, float ∗e2)

  get_monomer_free_energies

- void initialize_cofold (int length)

8.4.1 Detailed Description

MFE version of cofolding routines.

This file includes (almost) all function declarations within the **RNAlib** that are related to MFE Cofolding... This also includes the Zuker suboptimals calculations, since they are implemented using the cofold routines.

8.4.2 Function Documentation

8.4.2.1 void get_monomere_mfes (float $*$ *e1*, float $*$ *e2*)

get_monomer_free_energies

Export monomer free energies out of cofold arrays

Parameters:

e1 A pointer to a variable where the energy of molecule A will be written to

e2 A pointer to a variable where the energy of molecule B will be written to

8.4.2.2 void initialize_cofold (int *length*)

allocate arrays for folding

Deprecated

{This function is obsolete and will be removed soon!}

8.5 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/convert_epars.h File Reference

Functions and definitions for energy parameter file format conversion.

Defines

- #define VRNA_CONVERT_OUTPUT_ALL 1U
- #define VRNA_CONVERT_OUTPUT_HP 2U
- #define VRNA_CONVERT_OUTPUT_STACK 4U
- #define VRNA_CONVERT_OUTPUT_MM_HP 8U
- #define VRNA_CONVERT_OUTPUT_MM_INT 16U
- #define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U
- #define VRNA_CONVERT_OUTPUT_MM_INT_23 64U
- #define VRNA_CONVERT_OUTPUT_MM_MULTI 128U
- #define VRNA_CONVERT_OUTPUT_MM_EXT 256U
- #define VRNA_CONVERT_OUTPUT_DANGLE5 512U
- #define VRNA_CONVERT_OUTPUT_DANGLE3 1024U
- #define VRNA_CONVERT_OUTPUT_INT_11 2048U
- #define VRNA_CONVERT_OUTPUT_INT_21 4096U
- #define VRNA_CONVERT_OUTPUT_INT_22 8192U
- #define VRNA_CONVERT_OUTPUT_BULGE 16384U
- #define VRNA_CONVERT_OUTPUT_INT 32768U
- #define VRNA_CONVERT_OUTPUT_ML 65536U
- #define VRNA_CONVERT_OUTPUT_MISC 131072U
- #define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U
- #define VRNA_CONVERT_OUTPUT_VANILLA 524288U
- #define VRNA_CONVERT_OUTPUT_NINIO 1048576U
- #define VRNA_CONVERT_OUTPUT_DUMP 2097152U

Functions

- void convert_parameter_file (const char ∗iname, const char ∗oname, unsigned int options)

8.5.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

8.6 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/data_structures.h File Reference

All datastructures and typedefs shared among the Vienna RNA Package can be found here.

Include dependency graph for data_structures.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct plist

  this datastructure is used as input parameter in functions of PS_dot.h and others

- struct cpair

  this datastructure is used as input parameter in functions of PS_dot.c

- struct COORDINATE

  this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE

- struct sect

  Stack of partial structures for backtracking.

- struct bondT

  Base pair.

- struct bondTEn

  Base pair with associated energy.

- struct model_detailsT

  The data structure that contains the complete model details used throughout the calculations.

- struct paramT

  The datastructure that contains temperature scaled energy parameters.

- struct pf_paramT

  The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.

- struct PAIR

  Base pair data structure used in subopt.c.

- struct INTERVAL

  Sequence interval stack element used in subopt.c.

- struct SOLUTION

  Solution element from subopt.c.

- struct **cofoldF**
- struct **ConcEnt**
- struct **pairpro**
- struct pair_info

  A base pair info structure.

- struct **move**
- struct **intermediate**
- struct **path**
- struct pu_contrib

  contributions to p_u

- struct **interact**
- struct pu_out

  Collection of all free_energy of beeing unpaired values for output.

- struct constrain

  constraints for cofolding

- struct **duplexT**
- struct **node**
- struct **snoopT**
- struct **dupVar**
- struct TwoDfold_solution

  Solution element returned from TwoDfoldList.

- struct TwoDfold_vars

  Variables compound for 2Dfold MFE folding.

- struct TwoDpfold_solution

  Solution element returned from TwoDpfoldList.

- struct TwoDpfold_vars

  Variables compound for 2Dfold partition function folding.

Defines

- #define MAXALPHA 20

  Maximal length of alphabet.

- #define MAXDOS 1000

  Maximum density of states discretization for subopt.

8.6.1 Detailed Description

All datastructures and typedefs shared among the Vienna RNA Package can be found here.

8.7 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/dist_vars.h File Reference

Global variables for Distance-Package.

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **Postorder_list**
- struct **Tree**
- struct **swString**

Variables

- int edit_backtrack

  Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.

- char ∗ aligned_line [4]

  Contains the two aligned structures after a call to one of the distance functions with edit_backtrack set to 1.

- int cost_matrix

  Specify the cost matrix to be used for distance calculations.

8.7.1 Detailed Description

Global variables for Distance-Package.

8.7.2 Variable Documentation

8.7.2.1 int edit_backtrack

Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.

set to 1 if you want backtracking

8.7.2.2 int cost_matrix

Specify the cost matrix to be used for distance calculations.

if 0, use the default cost matrix (upper matrix in example), otherwise use Shapiro's costs (lower matrix).

8.8 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/duplex.h File Reference

Duplex folding function declarations...

Include dependency graph for duplex.h:

8.8.1 Detailed Description

Duplex folding function declarations...

8.9 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/edit_cost.h File Reference

global variables for Edit Costs included by treedist.c and stringdist.c

8.9.1 Detailed Description

global variables for Edit Costs included by treedist.c and stringdist.c

8.10 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/energy_const.h File Reference

Include dependency graph for energy_const.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define GASCONST 1.98717
- #define K0 273.15
- #define INF 10000000
- #define FORBIDDEN 9999
- #define BONUS 10000
- #define NBPAIRS 7
- #define TURN 3
- #define MAXLOOP 30

8.10.1 Detailed Description

energy constants

8.10.2 Define Documentation

8.10.2.1 #define GASCONST 1.98717

The gas constant

8.10.2.2 #define K0 273.15

0 deg Celsius in Kelvin

8.10.2.3 #define INF 10000000

Infinity as used in minimization routines

8.10.2.4 #define FORBIDDEN 9999

forbidden

8.10.2.5 #define BONUS 10000

bonus contribution

8.10.2.6 #define NBPAIRS 7

The number of distinguishable base pairs

8.10.2.7 #define TURN 3

The minimum loop length

8.10.2.8 #define MAXLOOP 30

The maximum loop length

8.11 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/findpath.h File Reference

Compute direct refolding paths between two secondary structures.

Include dependency graph for findpath.h:

Functions

- int find_saddle (const char ∗seq, const char ∗struc1, const char ∗struc2, int max)

  Find energy of a saddle point between 2 structures (serch only direct path).

- path_t ∗ get_path (const char ∗seq, const char ∗s1, const char ∗s2, int maxkeep)

  Find refolding path between 2 structures (serch only direct path).

- void free_path (path_t ∗path)

  Free memory allocated by get_path() function.

8.11.1 Detailed Description

Compute direct refolding paths between two secondary structures.

8.11.2 Function Documentation

8.11.2.1 int find_saddle (const char ∗ *seq*, const char ∗ *struc1*, const char ∗ *struc2*, int *max*)

Find energy of a saddle point between 2 structures (serch only direct path).

Parameters:

seq RNA sequence

struc1 A pointer to the character array where the first secondary structure in dot-bracket notation will be written to

struc2 A pointer to the character array where the second secondary structure in dot-bracket notation will be written to

max integer how many strutures are being kept during the search

Returns:

the saddle energy in 10cal/mol

8.11.2.2 path_t∗ get_path (const char ∗ *seq*, const char ∗ *s1*, const char ∗ *s2*, int *maxkeep*)

Find refolding path between 2 structures (serch only direct path).

Parameters:

- ***seq*** RNA sequence
- ***s1*** A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
- ***s2*** A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
- ***maxkeep*** integer how many strutures are being kept during the search

Returns:

direct refolding path between two structures

8.11.2.3 void free_path (path_t ∗ *path*)

Free memory allocated by get_path() function.

Parameters:

- ***path*** pointer to memory to be freed

8.12 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/fold.h File Reference

MFE calculations and energy evaluations for single RNA sequences.

Include dependency graph for fold.h:

Functions

- float fold_par (const char ∗sequence, char ∗structure, paramT ∗parameters, int is_constrained, int is_circular)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

- float fold (const char ∗sequence, char ∗structure)

  Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

- float circfold (const char ∗sequence, char ∗structure)

  Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.

- float energy_of_structure (const char ∗string, const char ∗structure, int verbosity_level)

  Calculate the free energy of an already folded RNA using global model detail settings.

- float energy_of_struct_par (const char ∗string, const char ∗structure, paramT ∗parameters, int verbosity_level)

  Calculate the free energy of an already folded RNA.

- float energy_of_circ_structure (const char ∗string, const char ∗structure, int verbosity_level)

  Calculate the free energy of an already folded circular RNA.

- float energy_of_circ_struct_par (const char ∗string, const char ∗structure, param-T ∗parameters, int verbosity_level)

  Calculate the free energy of an already folded circular RNA.

- int energy_of_structure_pt (const char ∗string, short ∗ptable, short ∗s, short ∗s1, int verbosity_level)

  Calculate the free energy of an already folded RNA.

- int energy_of_struct_pt_par (const char ∗string, short ∗ptable, short ∗s, short ∗s1, paramT ∗parameters, int verbosity_level)

  Calculate the free energy of an already folded RNA.

- void free_arrays (void)

Free arrays for mfe folding.

- void parenthesis_structure (char ∗structure, bondT ∗bp, int length)

  Create a dot-backet/parenthesis structure from backtracking stack.

- void parenthesis_zuker (char ∗structure, bondT ∗bp, int length)

  Create a dot-backet/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.

- void update_fold_params (void)

  Recalculate energy parameters.

- float energy_of_move (const char ∗string, const char ∗structure, int m1, int m2)

  Calculate energy of a move (closing or opening of a base pair).

- int energy_of_move_pt (short ∗pt, short ∗s, short ∗s1, int m1, int m2)

  Calculate energy of a move (closing or opening of a base pair).

- int loop_energy (short ∗ptable, short ∗s, short ∗s1, int i)

  Calculate energy of a loop.

- void assign_plist_from_db (plist ∗∗pl, const char ∗struc, float pr)

  Create a plist from a dot-bracket string.

- int LoopEnergy (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1)
- int HairpinE (int size, int type, int si1, int sj1, const char ∗string)
- void initialize_fold (int length)
- float energy_of_struct (const char ∗string, const char ∗structure)
- int energy_of_struct_pt (const char ∗string, short ∗ptable, short ∗s, short ∗s1)
- float energy_of_circ_struct (const char ∗string, const char ∗structure)

Variables

- int logML

  if nonzero use logarithmic ML energy in energy_of_struct

- int uniq_ML

  do ML decomposition uniquely (for subopt)

- int cut_point

  set to first pos of second seq for cofolding

- int eos_debug

  verbose info from energy_of_struct

8.12.1 Detailed Description

MFE calculations and energy evaluations for single RNA sequences.

This file includes (almost) all function declarations within the RNAlib that are related to MFE folding...

8.12.2 Function Documentation

8.12.2.1 void parenthesis_structure (char ∗ *structure*, bondT ∗ *bp*, int *length*)

Create a dot-backet/parenthesis structure from backtracking stack.

Note:

This function is threadsafe

8.12.2.2 void parenthesis_zuker (char ∗ *structure*, bondT ∗ *bp*, int *length*)

Create a dot-backet/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.

Note:

This function is threadsafe

8.12.2.3 float energy_of_move (const char ∗ *string*, const char ∗ *structure*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair).

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

See also:

make_pair_table(), energy_of_move()

Parameters:

string RNA sequence

structure secondary structure in dot-bracket notation

m1 first coordinate of base pair

m2 second coordinate of base pair

Returns:

energy change of the move in kcal/mol

8.12.2.4 int energy_of_move_pt (short ∗ *pt*, short ∗ *s*, short ∗ *s1*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair).

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

See also:

make_pair_table(), energy_of_move()

Parameters:

pt the pair table of the secondary structure

s encoded RNA sequence

s1 encoded RNA sequence

m1 first coordinate of base pair

m2 second coordinate of base pair

Returns:

energy change of the move in 10cal/mol

8.12.2.5 int loop_energy (short ∗ *ptable*, short ∗ *s*, short ∗ *s1*, int *i*)

Calculate energy of a loop.

Parameters:

ptable the pair table of the secondary structure

s encoded RNA sequence

s1 encoded RNA sequence

i position of covering base pair

Returns:

free energy of the loop in 10cal/mol

8.12.2.6 void assign_plist_from_db (plist ∗∗ *pl*, const char ∗ *struc*, float *pr*)

Create a plist from a dot-bracket string.

The dot-bracket string is parsed and for each base pair an entry in the plist is created. The probability of each pair in the list is set by a function parameter.

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

This function is threadsafe

Parameters:

pl A pointer to the plist that is to be created
struc The secondary structure in dot-bracket notation
pr The probability for each base pair

8.12.2.7 int LoopEnergy (int *n1*, int *n2*, int *type*, int *type_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*)

Deprecated

{This function is deprecated and will be removed soon. Use E_IntLoop() instead!}

8.12.2.8 int HairpinE (int *size*, int *type*, int *si1*, int *sj1*, const char ∗ *string*)

Deprecated

{This function is deprecated and will be removed soon. Use E_Hairpin() instead!}

8.12.2.9 void initialize_fold (int *length*)

Allocate arrays for folding

Deprecated

{This function is deprecated and will be removed soon!}

8.12.2.10 float energy_of_struct (const char ∗ *string*, const char ∗ *structure*)

Calculate the free energy of an already folded RNA

Note:

This function is not entirely threadsafe! Depending on the state of the global variable eos_debug it prints energy information to stdout or not...

Deprecated

This function is deprecated and should not be used in future programs! Use energy_of_structure() instead!

See also:

energy_of_structure, energy_of_circ_struct(), energy_of_struct_pt()

Parameters:

string RNA sequence

structure secondary structure in dot-bracket notation

Returns:

the free energy of the input structure given the input sequence in kcal/mol

8.12.2.11 int energy_of_struct_pt (const char ∗ *string*, short ∗ *ptable*, short ∗ *s*, short ∗ *s1*)

Calculate the free energy of an already folded RNA

Note:

This function is not entirely threadsafe! Depending on the state of the global variable eos_debug it prints energy information to stdout or not...

Deprecated

This function is deprecated and should not be used in future programs! Use energy_of_structure_pt() instead!

See also:

make_pair_table(), energy_of_structure()

Parameters:

string RNA sequence

ptable the pair table of the secondary structure

s encoded RNA sequence

s1 encoded RNA sequence

Returns:

the free energy of the input structure given the input sequence in 10kcal/mol

8.12.2.12 float energy_of_circ_struct (const char ∗ *string*, const char ∗ *structure*)

Calculate the free energy of an already folded circular RNA

Note:

This function is not entirely threadsafe! Depending on the state of the global variable eos_debug it prints energy information to stdout or not...

Deprecated

This function is deprecated and should not be used in future programs Use energy_of_circ_structure() instead!

See also:

energy_of_circ_structure(), energy_of_struct(), energy_of_struct_pt()

Parameters:

string RNA sequence

structure secondary structure in dot-bracket notation

Returns:

the free energy of the input structure given the input sequence in kcal/mol

8.13 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/fold_vars.h File Reference

Here all all declarations of the global variables used throughout RNAlib.

Include dependency graph for fold_vars.h:

This graph shows which files directly or indirectly include this file:

Functions

- void set_model_details (model_detailsT ∗md)

  Set default model details.

Variables

- int fold_constrained

  Global switch to activate/deactivate folding with structure constraints.

- int noLonelyPairs

  Global switch to avoid/allow helices of length 1.

- int dangles

  Switch the energy model for dangling end contributions (0, 1, 2, 3).

- int noGU

  Global switch to forbid/allow GU base pairs at all.

- int no_closingGU

  GU allowed only inside stacks if set to 1.

- int tetra_loop

  Include special stabilizing energies for some tri-, tetra- and hexa-loops;.

- int energy_set

  0 = BP; 1=any mit GC; 2=any mit AU-parameter

- int circ

  backward compatibility variable.. this does not effect anything

- int csv

  generate comma seperated output

- int oldAliEn
- int ribo

- char ∗ RibosumFile
- char ∗ nonstandards

  contains allowed non standard base pairs

- double temperature

  Rescale energy parameters to a temperature in degC.

- int james_rule
- int logML
- int cut_point

  Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.

- bondT ∗ base_pair

  Contains a list of base pairs after a call to fold().

- double ∗ pr

  A pointer to the base pair probability matrix.

- int ∗ iindx

  index array to move through pr.

- double pf_scale

  A scaling factor used by pf_fold() to avoid overflows.

- int do_backtrack

  do backtracking, i.e. compute secondary structures or base pair probabilities

- char backtrack_type

  A backtrack array marker for inverse_fold().

- int gquad

  Allow G-quadruplex formation.

8.13.1 Detailed Description

Here all all declarations of the global variables used throughout RNAlib.

8.13.2 Function Documentation

8.13.2.1 void set_model_details (model_detailsT ∗ *md*)

Set default model details.

Use this function if you wish to initialize a model_detailsT data structure with its default values, i.e. the global model settings

See also:

Parameters:

md A pointer to the data structure that shall be initialized

8.13.3 Variable Documentation

8.13.3.1 int noLonelyPairs

Global switch to avoid/allow helices of length 1.

Disallow all pairs which can only occur as lonely pairs (i.e. as helix of length 1). This avoids lonely base pairs in the predicted structures in most cases.

8.13.3.2 int dangles

Switch the energy model for dangling end contributions (0, 1, 2, 3).

If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (dangles = 1) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm pf_fold() these checks are neglected. If dangles is set to 2, all folding routines will follow this convention. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.

If dangles = 3 co-axial stacking is explicitly included for adjacent helices in mutliloops. The option affects only mfe folding and energy evaluation (fold() and energy_of_structure()), as well as suboptimal folding (subopt()) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

Default is 2 in most algorithms, partition function algorithms can only handle 0 and 2

8.13.3.3 int tetra_loop

Include special stabilizing energies for some tri-, tetra- and hexa-loops;.

default is 1.

8.13.3.4 int energy_set

0 = BP; 1=any mit GC; 2=any mit AU-parameter

If set to 1 or 2: fold sequences from an artificial alphabet ABCD..., where A pairs B, C pairs D, etc. using either GC (1) or AU parameters (2); default is 0, you probably don't want to change it.

8.13.3.5 int oldAliEn

use old alifold energies (with gaps)

8.13.3.6 int ribo

use ribosum matrices

8.13.3.7 char* RibosumFile

warning this variable will vanish in the future ribosums will be compiled in instead

8.13.3.8 char* nonstandards

contains allowed non standard base pairs

Lists additional base pairs that will be allowed to form in addition to GC, CG, AU, UA, GU and UG. Nonstandard base pairs are given a stacking energy of 0.

8.13.3.9 double temperature

Rescale energy parameters to a temperature in degC.

Default is 37C. You have to call the update_..._params() functions after changing this parameter.

8.13.3.10 int james_rule

interior loops of size 2 get energy 0.8Kcal and no mismatches, default 1

8.13.3.11 int logML

use logarithmic multiloop energy function

8.13.3.12 int cut_point

Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.

To evaluate the energy of a duplex structure (a structure formed by two strands), concatenate the to sequences and set it to the first base of the second strand in the concatenated sequence. The default value of -1 stands for single molecule folding. The cut_point variable is also used by PS_rna_plot() and PS_dot_plot() to mark the chain break in postscript plots.

8.13.3.13 bondT* base_pair

Contains a list of base pairs after a call to fold().

base_pair[0].i contains the total number of pairs.

Deprecated

Do not use this variable anymore!

8.13.3.14 double* pr

A pointer to the base pair probability matrix.

Deprecated

Do not use this variable anymore!

8.13.3.15 int* iindx

index array to move through pr.

The probability for base i and j to form a pair is in pr[iindx[i]-j].

Deprecated

Do not use this variable anymore!

8.13.3.16 double pf_scale

A scaling factor used by pf_fold() to avoid overflows.

Should be set to approximately $exp((-F/kT)/length)$, where F is an estimate for the ensemble free energy, for example the minimum free energy. You must call update_-pf_params() after changing this parameter.

If pf_scale is -1 (the default) , an estimate will be provided automatically when computing partition functions, e.g. pf_fold() The automatic estimate is usually insufficient for sequences more than a few hundred bases long.

8.13.3.17 int do_backtrack

do backtracking, i.e. compute secondary structures or base pair probabilities

If 0, do not calculate pair probabilities in pf_fold(); this is about twice as fast. Default is 1.

8.13.3.18 char backtrack_type

A backtrack array marker for inverse_fold().

If set to 'C': force (1,N) to be paired, 'M' fold as if the sequence were inside a multi-loop. Otherwise ('F') the usual mfe structure is computed.

8.14 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/gquad.h File Reference

Various functions related to G-quadruplex computations.

Include dependency graph for gquad.h:

Functions

- int ∗ get_gquad_matrix (short ∗S, paramT ∗P)

  Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.

- int parse_gquad (const char ∗struc, int ∗L, int l[3])
- PRIVATE int backtrack_GQuad_IntLoop (int c, int i, int j, int type, short ∗S, int ∗ggg, int ∗index, int ∗p, int ∗q, paramT ∗P)
- PRIVATE int backtrack_GQuad_IntLoop_L (int c, int i, int j, int type, short ∗S, int ∗∗ggg, int maxdist, int ∗p, int ∗q, paramT ∗P)

8.14.1 Detailed Description

Various functions related to G-quadruplex computations.

8.14.2 Function Documentation

8.14.2.1 int∗ get_gquad_matrix (short ∗ *S*, paramT ∗ *P*)

Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.

At each position ij in the matrix, the minimum free energy of any G-quadruplex delimited by i and j is stored. If no G-quadruplex formation is possible, the matrix element is set to INF. Access the elements in the matrix via matrix[indx[j]+i]. To get the integer array indx see get_jindx().

See also:

get_jindx(), encode_sequence()

Parameters:

S The encoded sequence

P A pointer to the data structure containing the precomputed energy contributions

Returns:

A pointer to the G-quadruplex contribution matrix

8.14.2.2 int parse_gquad (const char $*$ *struc*, int $*$ *L*, int *l*[3])

given a dot-bracket structure (possibly) containing gquads encoded by '+' signs, find first gquad, return end position or 0 if none found Upon return L and l[] contain the number of stacked layers, as well as the lengths of the linker regions. To parse a string with many gquads, call parse_gquad repeatedly e.g. end1 = parse_gquad(struc, &L, l); ... ; end2 = parse_gquad(struc+end1, &L, l); end2+=end1; ... ; end3 = parse_-gquad(struc+end2, &L, l); end3+=end2; ... ;

8.14.2.3 PRIVATE int backtrack_GQuad_IntLoop (int *c*, int *i*, int *j*, int *type*, short $*$ *S*, int $*$ *ggg*, int $*$ *index*, int $*$ *p*, int $*$ *q*, paramT $*$ *P*)

backtrack an interior loop like enclosed g-quadruplex with closing pair (i,j)

Parameters:

c The total contribution the loop should resemble

i position i of enclosing pair

j position j of enclosing pair

type base pair type of enclosing pair (must be reverse type)

S integer encoded sequence

ggg triangular matrix containing g-quadruplex contributions

index the index for accessing the triangular matrix

p here the 5' position of the gquad is stored

q here the 3' position of the gquad is stored

P the datastructure containing the precalculated contibutions

Returns:

1 on success, 0 if no gquad found

8.14.2.4 PRIVATE int backtrack_GQuad_IntLoop_L (int *c*, int *i*, int *j*, int *type*, short $*$ *S*, int $**$ *ggg*, int *maxdist*, int $*$ *p*, int $*$ *q*, paramT $*$ *P*)

backtrack an interior loop like enclosed g-quadruplex with closing pair (i,j) with underlying Lfold matrix

Parameters:

c The total contribution the loop should resemble

i position i of enclosing pair

j position j of enclosing pair

type base pair type of enclosing pair (must be reverse type)

S integer encoded sequence

ggg triangular matrix containing g-quadruplex contributions

p here the 5' position of the gquad is stored

q here the 3' position of the gquad is stored

P the datastructure containing the precalculated contibutions

Returns:

1 on success, 0 if no gquad found

8.15 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/inverse.h File Reference

Inverse folding routines.

Functions

- float inverse_fold (char ∗start, const char ∗target)

  Find sequences with predefined structure.

- float inverse_pf_fold (char ∗start, const char ∗target)

  Find sequence that maximizes probability of a predefined structure.

Variables

- char ∗ symbolset

  This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.

- float final_cost
- int give_up
- int inv_verbose

8.15.1 Detailed Description

Inverse folding routines.

8.16 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/Lfold.h File Reference

Predicting local MFE structures of large sequences.

Functions

- float Lfold (const char ∗string, char ∗structure, int maxdist)

  The local analog to fold().

- float Lfoldz (const char ∗string, char ∗structure, int maxdist, int zsc, double min_z)
- float aliLfold (const char ∗∗strings, char ∗structure, int maxdist)

8.16.1 Detailed Description

Predicting local MFE structures of large sequences.

8.17 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/loop_energies.h File Reference

Energy evaluation for MFE and partition function calculations.

Include dependency graph for loop_energies.h:

Functions

- PRIVATE int E_IntLoop (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, paramT ∗P)
- PRIVATE int E_Hairpin (int size, int type, int si1, int sj1, const char ∗string, paramT ∗P)
- PRIVATE int E_Stem (int type, int si1, int sj1, int extLoop, paramT ∗P)
- PRIVATE double exp_E_Stem (int type, int si1, int sj1, int extLoop, pf_paramT ∗P)
- PRIVATE double exp_E_Hairpin (int u, int type, short si1, short sj1, const char ∗string, pf_paramT ∗P)
- PRIVATE double exp_E_IntLoop (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, pf_paramT ∗P)

8.17.1 Detailed Description

Energy evaluation for MFE and partition function calculations.

This file contains functions for the calculation of the free energy ΔG of a hairpin- [E_Hairpin()] or interior-loop [E_IntLoop()] .

The unit of the free energy returned is $10^{-2} * \mathrm{kcal/mol}$

In case of computing the partition function, this file also supplies functions which return the Boltzmann weights $e^{-\Delta G/kT}$ for a hairpin- [exp_E_Hairpin()] or interior-loop [exp_E_IntLoop()].

8.17.2 Function Documentation

8.17.2.1 PRIVATE int E_IntLoop (int *n1*, int *n2*, int *type*, int *type_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*, paramT ∗ *P*)

Compute the Energy of an interior-loop

This function computes the free energy ΔG of an interior-loop with the following structure:

```
        3'  5'
        |   |
        U - V
    a_n       b_1
```

```
        .          .
        .          .
        .          .
       a_1        b_m
           X - Y
           |   |
           5'  3'
```

This general structure depicts an interior-loop that is closed by the base pair (X,Y). The enclosed base pair is (V,U) which leaves the unpaired bases a_1-a_n and b_1-b_n that constitute the loop. In this example, the length of the interior-loop is $(n+m)$ where n or m may be 0 resulting in a bulge-loop or base pair stack. The mismatching nucleotides for the closing pair (X,Y) are:

5'-mismatch: a_1

3'-mismatch: b_m

and for the enclosed base pair (V,U):

5'-mismatch: b_1

3'-mismatch: a_n

Note:

Base pairs are always denoted in 5'->3' direction. Thus the enclosed base pair must be 'turned arround' when evaluating the free energy of the interior-loop

See also:

scale_parameters()
paramT

Note:

This function is threadsafe

Parameters:

n1 The size of the 'left'-loop (number of unpaired nucleotides)
n2 The size of the 'right'-loop (number of unpaired nucleotides)
type The pair type of the base pair closing the interior loop
type_2 The pair type of the enclosed base pair
si1 The 5'-mismatching nucleotide of the closing pair
sj1 The 3'-mismatching nucleotide of the closing pair
sp1 The 3'-mismatching nucleotide of the enclosed pair
sq1 The 5'-mismatching nucleotide of the enclosed pair
P The datastructure containing scaled energy parameters

Returns:

The Free energy of the Interior-loop in dcal/mol

8.17.2.2 PRIVATE int E_Hairpin (int *size*, int *type*, int *si1*, int *sj1*, const char * *string*, paramT * *P*)

Compute the Energy of a hairpin-loop

To evaluate the free energy of a hairpin-loop, several parameters have to be known. A general hairpin-loop has this structure:

```
    a3 a4
  a2     a5
  a1     a6
    X - Y
    |   |
    5'  3'
```

where X-Y marks the closing pair [e.g. a **(G,C)** pair]. The length of this loop is 6 as there are six unpaired nucleotides (a1-a6) enclosed by (X,Y). The 5' mismatching nucleotide is a1 while the 3' mismatch is a6. The nucleotide sequence of this loop is "a1.a2.a3.a4.a5.a6"

Note:

The parameter sequence should contain the sequence of the loop in capital letters of the nucleic acid alphabet if the loop size is below 7. This is useful for unusually stable tri-, tetra- and hexa-loops which are treated differently (based on experimental data) if they are tabulated.

See also:

scale_parameters()
paramT

Warning:

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!
Energy evaluation may change due to updates in global variable "tetra_loop"

Parameters:

size The size of the loop (number of unpaired nucleotides)

type The pair type of the base pair closing the hairpin

si1 The 5'-mismatching nucleotide

sj1 The 3'-mismatching nucleotide

string The sequence of the loop

P The datastructure containing scaled energy parameters

Returns:

The Free energy of the Hairpin-loop in dcal/mol

8.17.2.3 PRIVATE int E_Stem (int *type*, int *si1*, int *sj1*, int *extLoop*, paramT * *P*)

Compute the energy contribution of a stem branching off a loop-region

This function computes the energy contribution of a stem that branches off a loop region. This can be the case in multiloops, when a stem branching off increases the degree of the loop but also *immediately interior base pairs* of an exterior loop contribute free energy. To switch the bahavior of the function according to the evaluation of a multiloop- or exterior-loop-stem, you pass the flag 'extLoop'. The returned energy contribution consists of a TerminalAU penalty if the pair type is greater than 2, dangling end contributions of mismatching nucleotides adjacent to the stem if only one of the si1, sj1 parameters is greater than 0 and mismatch energies if both mismatching nucleotides are positive values. Thus, to avoid incooperating dangling end or mismatch energies just pass a negative number, e.g. -1 to the mismatch argument.

This is an illustration of how the energy contribution is assembled:

```
        3'  5'
        |   |
        X - Y
  5'-si1     sj1-3'
```

Here, (X,Y) is the base pair that closes the stem that branches off a loop region. The nucleotides si1 and sj1 are the 5'- and 3'- mismatches, respectively. If the base pair type of (X,Y) is greater than 2 (i.e. an A-U or G-U pair, the TerminalAU penalty will be included in the energy contribution returned. If si1 and sj1 are both nonnegative numbers, mismatch energies will also be included. If one of sij or sj1 is a negtive value, only 5' or 3' dangling end contributions are taken into account. To prohibit any of these mismatch contributions to be incoorporated, just pass a negative number to both, si1 and sj1. In case the argument extLoop is 0, the returned energy contribution also includes the *internal-loop-penalty* of a multiloop stem with closing pair type.

See also:

E_MLstem()
E_ExtLoop()

Note:

This function is threadsafe

Parameters:

type The pair type of the first base pair un the stem

si1 The 5'-mismatching nucleotide

sj1 The 3'-mismatching nucleotide

extLoop A flag that indicates whether the contribution reflects the one of an exterior loop or not

P The datastructure containing scaled energy parameters

Returns:

The Free energy of the branch off the loop in dcal/mol

8.17.2.4 PRIVATE double exp_E_Stem (int *type*, int *si1*, int *sj1*, int *extLoop*, pf_paramT ∗ *P*)

Compute the Boltzmann weighted energy contribution of a stem branching off a loop-region

This is the partition function variant of E_Stem()

See also:

E_Stem()

Note:

This function is threadsafe

Returns:

The Boltzmann weighted energy contribution of the branch off the loop

8.17.2.5 PRIVATE double exp_E_Hairpin (int *u*, int *type*, short *si1*, short *sj1*, const char ∗ *string*, pf_paramT ∗ *P*)

Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop

multiply by scale[u+2]

See also:

get_scaled_pf_parameters()
pf_paramT
E_Hairpin()

Warning:

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!
Energy evaluation may change due to updates in global variable "tetra_loop"

Parameters:

u The size of the loop (number of unpaired nucleotides)

type The pair type of the base pair closing the hairpin

si1 The 5'-mismatching nucleotide

sj1 The 3'-mismatching nucleotide

string The sequence of the loop

P The datastructure containing scaled Boltzmann weights of the energy parameters

Returns:

The Boltzmann weight of the Hairpin-loop

8.17.2.6 PRIVATE double exp_E_IntLoop (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*, pf_paramT ∗ *P*)

Compute Boltzmann weight $e^{-\Delta G/kT}$ of interior loop

multiply by scale[u1+u2+2] for scaling

See also:

get_scaled_pf_parameters()
pf_paramT
E_IntLoop()

Note:

This function is threadsafe

Parameters:

u1 The size of the 'left'-loop (number of unpaired nucleotides)

u2 The size of the 'right'-loop (number of unpaired nucleotides)

type The pair type of the base pair closing the interior loop

type2 The pair type of the enclosed base pair

si1 The 5'-mismatching nucleotide of the closing pair

sj1 The 3'-mismatching nucleotide of the closing pair

sp1 The 3'-mismatching nucleotide of the enclosed pair

sq1 The 5'-mismatching nucleotide of the enclosed pair

P The datastructure containing scaled Boltzmann weights of the energy parameters

Returns:

The Boltzmann weight of the Interior-loop

8.18 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/LPfold.h File Reference

Function declarations of partition function variants of the Lfold algorithm.

Include dependency graph for LPfold.h:

Functions

- void update_pf_paramsLP (int length)
- plist ∗ pfl_fold (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, struct plist ∗∗dpp2, FILE ∗pUfp, FILE ∗spup)

    Compute partition functions for locally stable secondary structures.

- plist ∗ pfl_fold_par (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, struct plist ∗∗dpp2, FILE ∗pUfp, FILE ∗spup, pf_paramT ∗parameters)

    Compute partition functions for locally stable secondary structures.

- void putoutpU_prob (double ∗∗pU, int length, int ulength, FILE ∗fp, int energies)

    Writes the unpaired probabilities (pU) or opening energies into a file.

- void putoutpU_prob_bin (double ∗∗pU, int length, int ulength, FILE ∗fp, int energies)

    Writes the unpaired probabilities (pU) or opening energies into a binary file.

- void init_pf_foldLP (int length)

8.18.1 Detailed Description

Function declarations of partition function variants of the Lfold algorithm.

8.18.2 Function Documentation

8.18.2.1 void init_pf_foldLP (int *length*)

Dunno if this function was ever used by external programs linking to RNAlib, but it was declared PUBLIC before. Anyway, never use this function as it will be removed soon and does nothing at all

8.19 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/MEA.h File Reference

Computes a MEA (maximum expected accuracy) structure.

Include dependency graph for MEA.h:

Functions

- float MEA (plist ∗p, char ∗structure, double gamma)

  Computes a MEA (maximum expected accuracy) structure.

8.19.1 Detailed Description

Computes a MEA (maximum expected accuracy) structure.

8.19.2 Function Documentation

8.19.2.1 float MEA (plist ∗ *p*, char ∗ *structure*, double *gamma*)

Computes a MEA (maximum expected accuracy) structure.

The algorithm maximizes the expected accuracy

$$A(S) = \sum_{(i,j) \in S} 2 \gamma p_{ij} + \sum_{i \notin S} p^u_i$$

Higher values of γ result in more base pairs of lower probability and thus higher sensitivity. Low values of γ result in structures containing only highly likely pairs (high specificity). The code of the MEA function also demonstrates the use of sparse dynamic programming scheme to reduce the time and memory complexity of folding.

8.20 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/mm.h File Reference

Several Maximum Matching implementations.

8.20.1 Detailed Description

Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

8.21 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/naview.h File Reference

This graph shows which files directly or indirectly include this file:

8.21.1 Detailed Description

8.22 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/params.h File Reference

Include dependency graph for params.h:

This graph shows which files directly or indirectly include this file:

Functions

- paramT ∗ scale_parameters (void)

  Get precomputed energy contributions for all the known loop types.

- paramT ∗ get_scaled_parameters (double temperature, model_detailsT md)

  Get precomputed energy contributions for all the known loop types.

- pf_paramT ∗ get_scaled_pf_parameters (void)
- pf_paramT ∗ get_boltzmann_factors (double temperature, double betaScale, model_detailsT md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.

- pf_paramT ∗ get_boltzmann_factor_copy (pf_paramT ∗parameters)

  Get a copy of already precomputed Boltzmann factors.

- pf_paramT ∗ get_scaled_alipf_parameters (unsigned int n_seq)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant).

- PUBLIC pf_paramT ∗ get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, model_detailsT md, double pf_scale)

  Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.

8.22.1 Detailed Description

8.23 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/part_func.h File Reference

Partition function of single RNA sequences.

Include dependency graph for part_func.h:

Functions

- float pf_fold_par (const char ∗sequence, char ∗structure, pf_paramT ∗parameters, int calculate_bppm, int is_constrained, int is_circular)

  Compute the partition function Q for a given RNA sequence.

- float pf_fold (const char ∗sequence, char ∗structure)

  Compute the partition function Q of an RNA sequence.

- float pf_circ_fold (const char ∗sequence, char ∗structure)

  Compute the partition function of a circular RNA sequence.

- char ∗ pbacktrack (char ∗sequence)

  Sample a secondary structure from the Boltzmann ensemble according its probability .

- char ∗ pbacktrack_circ (char ∗sequence)

  Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

- void free_pf_arrays (void)

  Free arrays for the partition function recursions.

- void update_pf_params (int length)

  Recalculate energy parameters.

- void update_pf_params_par (int length, pf_paramT ∗parameters)

  Recalculate energy parameters.

- double ∗ export_bppm (void)

  Get a pointer to the base pair probability array
  Accessing the base pair probabilities for a pair (i,j) is achieved by.

- void assign_plist_from_pr (plist ∗∗pl, double ∗probs, int length, double cutoff)

  Create a plist from a probability matrix.

- int get_pf_arrays (short ∗∗S_p, short ∗∗S1_p, char ∗∗ptype_p, double ∗∗qb_p, double ∗∗qm_p, double ∗∗q1k_p, double ∗∗qln_p)

  Get the pointers to (almost) all relavant computation arrays used in partition function computation.

- double get_subseq_F (int i, int j)

  Get the free energy of a subsequence from the q[] array.

- char ∗ get_centroid_struct_pl (int length, double ∗dist, plist ∗pl)

  Get the centroid structure of the ensemble.

- char ∗ get_centroid_struct_pr (int length, double ∗dist, double ∗pr)

  Get the centroid structure of the ensemble.

- double mean_bp_distance (int length)

  Get the mean base pair distance of the last partition function computation.

- double mean_bp_distance_pr (int length, double ∗pr)

  Get the mean base pair distance in the thermodynamic ensemble.

- void bppm_to_structure (char ∗structure, double ∗pr, unsigned int length)

  Create a dot-bracket like structure string from base pair probability matrix.

- char bppm_symbol (const float ∗x)

  Get a pseudo dot bracket notation for a given probability information.

- void init_pf_fold (int length)

  Allocate space for pf_fold().

- char ∗ centroid (int length, double ∗dist)
- double mean_bp_dist (int length)
- double expLoopEnergy (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)
- double expHairpinEnergy (int u, int type, short si1, short sj1, const char ∗string)

Variables

- int st_back

  Flag indicating that auxilary arrays are needed throughout the computations. This is essential for stochastic backtracking.

8.23.1 Detailed Description

Partition function of single RNA sequences.

This file includes (almost) all function declarations within the **RNAlib** that are related to Partion function folding...

8.23.2 Function Documentation

8.23.2.1 void init_pf_fold (int *length*)

Allocate space for pf_fold().

Deprecated

This function is obsolete and will be removed soon!

8.23.2.2 char∗ centroid (int *length*, double ∗ *dist*)

Deprecated

This function is deprecated and should not be used anymore as it is not threadsafe!

See also:

get_centroid_struct_pl(), get_centroid_struct_pr()

8.23.2.3 double mean_bp_dist (int *length*)

get the mean pair distance of ensemble

Deprecated

This function is not threadsafe and should not be used anymore. Use mean_bp_-distance() instead!

8.23.2.4 double expLoopEnergy (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*)

Deprecated

Use exp_E_IntLoop() from loop_energies.h instead

8.23.2.5 double expHairpinEnergy (int *u*, int *type*, short *si1*, short *sj1*, const char ∗ *string*)

Deprecated

Use exp_E_Hairpin() from loop_energies.h instead

8.24 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/part_func_co.h File Reference

Partition function for two RNA sequences.

Include dependency graph for part_func_co.h:

Functions

- cofoldF co_pf_fold (char ∗sequence, char ∗structure)

  Calculate partition function and base pair probabilities.

- cofoldF co_pf_fold_par (char ∗sequence, char ∗structure, pf_paramT ∗parameters, int calculate_bppm, int is_constrained)

  Calculate partition function and base pair probabilities.

- double ∗ export_co_bppm (void)

  Get a pointer to the base pair probability array.

- void free_co_pf_arrays (void)

  Free the memory occupied by co_pf_fold().

- void update_co_pf_params (int length)

  Recalculate energy parameters.

- void update_co_pf_params_par (int length, pf_paramT ∗parameters)

  Recalculate energy parameters.

- void compute_probabilities (double FAB, double FEA, double FEB, struct plist ∗prAB, struct plist ∗prA, struct plist ∗prB, int Alength)

  Compute Boltzmann probabilities of dimerization without homodimers.

- ConcEnt ∗ get_concentrations (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double ∗startconc)

  Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

- plist ∗ get_plist (struct plist ∗pl, int length, double cut_off)
- void init_co_pf_fold (int length)

Variables

- int mirnatog

  Toggles no intrabp in 2nd mol.

- double F_monomer [2]

Free energies of the two monomers.

8.24.1 Detailed Description

Partition function for two RNA sequences.

As for folding one RNA molecule, this computes the partition function of all possible structures and the base pair probabilities. Uses the same global pf_scale variable to avoid overflows.

To simplify the implementation the partition function computation is done internally in a null model that does not include the duplex initiation energy, i.e. the entropic penalty for producing a dimer from two monomers). The resulting free energies and pair probabilities are initially relative to that null model. In a second step the free energies can be corrected to include the dimerization penalty, and the pair probabilities can be divided into the conditional pair probabilities given that a re dimer is formed or not formed.

After computing the partition functions of all possible dimeres one can compute the probabilities of base pairs, the concentrations out of start concentrations and sofar and soaway.

Dimer formation is inherently concentration dependent. Given the free energies of the monomers A and B and dimers AB, AA, and BB one can compute the equilibrium concentrations, given input concentrations of A and B, see e.g. Dimitrov & Zuker (2004)

8.24.2 Function Documentation

8.24.2.1 plist∗ get_plist (struct plist ∗ *pl*, int *length*, double *cut_off*)

DO NOT USE THIS FUNCTION ANYMORE

Deprecated

{ This function is deprecated and will be removed soon!} use assign_plist_from_-pr() instead!

8.24.2.2 void init_co_pf_fold (int *length*)

DO NOT USE THIS FUNCTION ANYMORE

Deprecated

{ This function is deprecated and will be removed soon!}

8.25 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/part_func_up.h File Reference

Partition Function Cofolding as stepwise process.

Include dependency graph for part_func_up.h:

Functions

- pu_contrib ∗ pf_unstru (char ∗sequence, int max_w)

  Calculate the partition function over all unpaired regions of a maximal length.

- interact ∗ pf_interact (const char ∗s1, const char ∗s2, pu_contrib ∗p_c, pu_contrib ∗p_c2, int max_w, char ∗cstruc, int incr3, int incr5)

  Calculates the probability of a local interaction between two sequences.

- void free_interact (interact ∗pin)

  Frees the output of function pf_interact().

- void free_pu_contrib_struct (pu_contrib ∗pu)

  Frees the output of function pf_unstru().

8.25.1 Detailed Description

Partition Function Cofolding as stepwise process.

In this approach to cofolding the interaction between two RNA molecules is seen as a stepwise process. In a first step, the target molecule has to adopt a structure in which a binding site is accessible. In a second step, the ligand molecule will hybridize with a region accessible to an interaction. Consequently the algorithm is designed as a two step process: The first step is the calculation of the probability that a region within the target is unpaired, or equivalently, the calculation of the free energy needed to expose a region. In the second step we compute the free energy of an interaction for every possible binding site.

8.26 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/plot_layouts.h File Reference

Secondary structure plot layout algorithms.

Include dependency graph for plot_layouts.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define VRNA_PLOT_TYPE_SIMPLE 0

  Definition of Plot type simple.

- #define VRNA_PLOT_TYPE_NAVIEW 1

  Definition of Plot type Naview.

- #define VRNA_PLOT_TYPE_CIRCULAR 2

  Definition of Plot type Circular.

Functions

- int simple_xy_coordinates (short ∗pair_table, float ∗X, float ∗Y)

  Calculate nucleotide coordinates for secondary structure plot the Simple way.

- int simple_circplot_coordinates (short ∗pair_table, float ∗x, float ∗y)

  Calculate nucleotide coordinates for Circular Plot.

Variables

- int rna_plot_type

  Switch for changing the secondary structure layout algorithm.

8.26.1 Detailed Description

Secondary structure plot layout algorithms.

c Ronny Lorenz The ViennaRNA Package

8.26.2 Define Documentation

8.26.2.1 #define VRNA_PLOT_TYPE_SIMPLE 0

Definition of Plot type *simple*.

This is the plot type definition for several RNA structure plotting functions telling them to use **Simple** plotting algorithm

See also:

rna_plot_type, PS_rna_plot_a(), PS_rna_plot(), svg_rna_plot(), gmlRNA(), ssv_-rna_plot(), xrna_plot()

8.26.2.2 #define VRNA_PLOT_TYPE_NAVIEW 1

Definition of Plot type *Naview*.

This is the plot type definition for several RNA structure plotting functions telling them to use **Naview** plotting algorithm

See also:

rna_plot_type, PS_rna_plot_a(), PS_rna_plot(), svg_rna_plot(), gmlRNA(), ssv_-rna_plot(), xrna_plot()

8.26.2.3 #define VRNA_PLOT_TYPE_CIRCULAR 2

Definition of Plot type *Circular*.

This is the plot type definition for several RNA structure plotting functions telling them to produce a **Circular plot**

See also:

rna_plot_type, PS_rna_plot_a(), PS_rna_plot(), svg_rna_plot(), gmlRNA(), ssv_-rna_plot(), xrna_plot()

8.26.3 Function Documentation

8.26.3.1 int simple_xy_coordinates (short ∗ *pair_table*, float ∗ *X*, float ∗ *Y*)

Calculate nucleotide coordinates for secondary structure plot the *Simple way*.

See also:

make_pair_table(), rna_plot_type, simple_circplot_coordinates(), naview_xy_-coordinates(), PS_rna_plot_a(), PS_rna_plot, svg_rna_plot()

Parameters:

pair_table The pair table of the secondary structure

X a pointer to an array with enough allocated space to hold the x coordinates

Y a pointer to an array with enough allocated space to hold the y coordinates

Returns:

length of sequence on success, 0 otherwise

8.26.3.2 int simple_circplot_coordinates (short ∗ *pair_table*, float ∗ *x*, float ∗ *y*)

Calculate nucleotide coordinates for *Circular Plot*.

This function calculates the coordinates of nucleotides mapped in equal distancies onto a unit circle.

Note:

In order to draw nice arcs using quadratic bezier curves that connect base pairs one may calculate a second tangential point P^t in addition to the actual R^2 coordinates. the simplest way to do so may be to compute a radius scaling factor rs in the interval $[0, 1]$ that weights the proportion of base pair span to the actual length of the sequence. This scaling factor can then be used to calculate the coordinates for P^t, i.e. $P^t_x[i] = X[i] * rs$ and $P^t_y[i] = Y[i] * rs$.

See also:

make_pair_table(), rna_plot_type, simple_xy_coordinates(), naview_xy_-coordinates(), PS_rna_plot_a(), PS_rna_plot, svg_rna_plot()

Parameters:

pair_table The pair table of the secondary structure

x a pointer to an array with enough allocated space to hold the x coordinates

y a pointer to an array with enough allocated space to hold the y coordinates

Returns:

length of sequence on success, 0 otherwise

8.26.4 Variable Documentation

8.26.4.1 int rna_plot_type

Switch for changing the secondary structure layout algorithm.

Current possibility are 0 for a simple radial drawing or 1 for the modified radial drawing taken from the *naview* program of Bruccoleri & Heinrich (1988).

Note:

To provide thread safety please do not rely on this global variable in future implementations but pass a plot type flag directly to the function that decides which layout algorithm it may use!

See also:

VRNA_PLOT_TYPE_SIMPLE, VRNA_PLOT_TYPE_NAVIEW, VRNA_-PLOT_TYPE_CIRCULAR

8.27 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/profiledist.h File Reference

Include dependency graph for profiledist.h:

Functions

- float profile_edit_distance (const float ∗T1, const float ∗T2)

  Align the 2 probability profiles T1, T2

  .
- float ∗ Make_bp_profile_bppm (double ∗bppm, int length)

  condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.
- void print_bppm (const float ∗T)

  print string representation of probability profile
- void free_profile (float ∗T)

  free space allocated in Make_bp_profile
- float ∗ Make_bp_profile (int length)

8.27.1 Detailed Description

8.27.2 Function Documentation

8.27.2.1 float profile_edit_distance (const float ∗ *T1*, const float ∗ *T2*)

Align the 2 probability profiles T1, T2

.

This is like a Needleman-Wunsch alignment, we should really use affine gap-costs ala Gotoh

8.27.2.2 float∗ Make_bp_profile_bppm (double ∗ *bppm*, int *length*)

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

This resulting probability profile is used as input for profile_edit_distance

Parameters:

bppm A pointer to the base pair probability matrix

length The length of the sequence

Returns:

The bp profile

8.27.2.3 void free_profile (float ∗ *T*)

free space allocated in Make_bp_profile

Backward compatibility only. You can just use plain free()

8.27.2.4 float∗ Make_bp_profile (int *length*)

Note:

This function is NOT threadsafe

See also:

Make_bp_profile_bppm()

Deprecated

This function is deprecated and will be removed soon! See Make_bp_profile_-bppm() for a replacement

8.28 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/PS_dot.h File Reference

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Include dependency graph for PS_dot.h:

Functions

- int PS_rna_plot (char ∗string, char ∗structure, char ∗file)

  Produce a secondary structure graph in PostScript and write it to 'filename'.

- int PS_rna_plot_a (char ∗string, char ∗structure, char ∗file, char ∗pre, char ∗post)

  Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

- int gmlRNA (char ∗string, char ∗structure, char ∗ssfile, char option)

  Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.

- int ssv_rna_plot (char ∗string, char ∗structure, char ∗ssfile)

  Produce a secondary structure graph in SStructView format.

- int svg_rna_plot (char ∗string, char ∗structure, char ∗ssfile)

  Produce a secondary structure plot in SVG format and write it to a file.

- int xrna_plot (char ∗string, char ∗structure, char ∗ssfile)

  Produce a secondary structure plot for further editing in XRNA.

- int PS_dot_plot_list (char ∗seq, char ∗filename, plist ∗pl, plist ∗mf, char ∗comment)

  Produce a postscript dot-plot from two pair lists.

- int aliPS_color_aln (const char ∗structure, const char ∗filename, const char ∗seqs[ ], const char ∗names[ ])
- int PS_dot_plot (char ∗string, char ∗file)

  Produce postscript dot-plot.

8.28.1 Detailed Description

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

8.28.2 Function Documentation

8.28.2.1 int PS_rna_plot (char ∗ *string*, char ∗ *structure*, char ∗ *file*)

Produce a secondary structure graph in PostScript and write it to 'filename'.

Note that this function has changed from previous versions and now expects the structure to be plotted in dot-bracket notation as an argument. It does not make use of the global base_pair array anymore.

Parameters:

string The RNA sequence

structure The secondary structure in dot-bracket notation

file The filename of the postscript output

Returns:

1 on success, 0 otherwise

8.28.2.2 int PS_rna_plot_a (char ∗ *string*, char ∗ *structure*, char ∗ *file*, char ∗ *pre*, char ∗ *post*)

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

Same as PS_rna_plot() but adds extra PostScript macros for various annotations (see generated PS code). The 'pre' and 'post' variables contain PostScript code that is verbatim copied in the resulting PS file just before and after the structure plot. If both arguments ('pre' and 'post') are NULL, no additional macros will be printed into the PostScript.

Parameters:

string The RNA sequence

structure The secondary structure in dot-bracket notation

file The filename of the postscript output

pre PostScript code to appear before the secondary structure plot

post PostScript code to appear after the secondary structure plot

Returns:

1 on success, 0 otherwise

8.28.2.3 int gmlRNA (char ∗ *string*, char ∗ *structure*, char ∗ *ssfile*, char *option*)

Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.

If 'option' is an uppercase letter the RNA sequence is used to label nodes, if 'option' equals *'X'* or *'x'* the resulting file will coordinates for an initial layout of the graph.

Parameters:

string The RNA sequence

structure The secondary structure in dot-bracket notation

ssfile The filename of the gml output

option The option flag

Returns:

1 on success, 0 otherwise

8.28.2.4 int ssv_rna_plot (char ∗ *string*, char ∗ *structure*, char ∗ *ssfile*)

Produce a secondary structure graph in SStructView format.

Write coord file for SStructView

Parameters:

string The RNA sequence

structure The secondary structure in dot-bracket notation

ssfile The filename of the ssv output

Returns:

1 on success, 0 otherwise

8.28.2.5 int svg_rna_plot (char ∗ *string*, char ∗ *structure*, char ∗ *ssfile*)

Produce a secondary structure plot in SVG format and write it to a file.

Parameters:

string The RNA sequence

structure The secondary structure in dot-bracket notation

ssfile The filename of the svg output

Returns:

1 on success, 0 otherwise

8.28.2.6 int xrna_plot (char ∗ *string*, char ∗ *structure*, char ∗ *ssfile*)

Produce a secondary structure plot for further editing in XRNA.

Parameters:

string The RNA sequence

structure The secondary structure in dot-bracket notation

ssfile The filename of the xrna output

Returns:

1 on success, 0 otherwise

8.28.2.7 int PS_dot_plot_list (char ∗ *seq*, char ∗ *filename*, plist ∗ *pl*, plist ∗ *mf*, char ∗ *comment*)

Produce a postscript dot-plot from two pair lists.

This function reads two plist structures (e.g. base pair probabilities and a secondary structure) as produced by assign_plist_from_pr() and assign_plist_from_db() and produces a postscript "dot plot" that is written to ’filename’.

Using base pair probabilities in the first and mfe structure in the second plist, the resulting "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy structure.

See also:

assign_plist_from_pr(), assign_plist_from_db()

Parameters:

seq The RNA sequence

filename A filename for the postscript output

pl The base pair probability pairlist

mf The mfe secondary structure pairlist

comment A comment

Returns:

1 if postscript was successfully written, 0 otherwise

8.28.2.8 int aliPS_color_aln (const char ∗ *structure*, const char ∗ *filename*, const char ∗ *seqs*[ ], const char ∗ *names*[ ])

PS_color_aln for duplexes

8.28.2.9 int PS_dot_plot (char ∗ *string*, char ∗ *file*)

Produce postscript dot-plot.

Wrapper to PS_dot_plot_list

Reads base pair probabilities produced by pf_fold() from the global array pr and the pair list base_pair produced by fold() and produces a postscript "dot plot" that is written to 'filename'. The "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy

Note:

DO NOT USE THIS FUNCTION ANYMORE SINCE IT IS NOT THREAD-SAFE

Deprecated

This function is deprecated and will be removed soon! Use PS_dot_plot_list() instead!

8.29 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/read_epars.h File Reference

Functions

- void read_parameter_file (const char fname[ ])

  Read energy parameters from a file.

- void write_parameter_file (const char fname[ ])

  Write energy parameters to a file.

8.29.1 Detailed Description

8.30 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/RNAstruct.h File Reference

Parsing and Coarse Graining of Structures.

Functions

- char ∗ b2HIT (const char ∗structure)

  Converts the full structure from bracket notation to the HIT notation including root.

- char ∗ b2C (const char ∗structure)

  Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

- char ∗ b2Shapiro (const char ∗structure)

  Converts the full structure from bracket notation to the weighted *coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.*

- char ∗ add_root (const char ∗structure)

  Adds a root to an un-rooted tree in any except bracket notation.

- char ∗ expand_Shapiro (const char ∗coarse)

  Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

- char ∗ expand_Full (const char ∗structure)

  Convert the full structure from bracket notation to the expanded notation including root.

- char ∗ unexpand_Full (const char ∗ffull)

  Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

- char ∗ unweight (const char ∗wcoarse)

  Strip weights from any weighted tree.

- void unexpand_aligned_F (char ∗align[2])

  Converts two aligned structures in expanded notation.

- void parse_structure (const char ∗structure)

  Collects a statistic of structure elements of the full structure in bracket notation.

Variables

- int loop_size [STRUC]

  contains a list of all loop sizes. loop_size[0] contains the number of external bases.

- int helix_size [STRUC]

  contains a list of all stack sizes.

- int loop_degree [STRUC]

  contains the corresponding list of loop degrees.

- int loops

  contains the number of loops (and therefore of stacks).

- int unpaired

  contains the number of unpaired bases.

- int pairs

  contains the number of base pairs in the last parsed structure.

8.30.1 Detailed Description

Parsing and Coarse Graining of Structures.

Example:

```
*   .((..(((...)))..((..)))).   is the bracket or full tree
*   becomes expanded:   - expand_Full() -
*   ((U)(((U)(U)((((U)(U)(U)P)P)P)(U)(U)(((U)(U)P)P)P)P)(U)R)
*   HIT:                - b2HIT() -
*   ((U1)((U2)((U3)P3)(U2)((U2)P2)P2)(U1)R)
*   Coarse:             - b2C() -
*   ((H)((H)M)R)
*   becomes expanded:   - expand_Shapiro() -
*   (((((H)S)((H)S)M)S)R)
*   weighted Shapiro:   - b2Shapiro() -
*   ((((((H3)S3)((H2)S2)M4)S2)E2)R)
*
```

8.30.2 Function Documentation

8.30.2.1 char∗ b2HIT (const char ∗ *structure*)

Converts the full structure from bracket notation to the HIT notation including root.

Parameters:

structure

Returns:

8.30.2.2 char∗ b2C (const char ∗ *structure*)

Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

Parameters:

structure

Returns:

8.30.2.3 char∗ b2Shapiro (const char ∗ *structure*)

Converts the full structure from bracket notation to the *weighted* coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.

Parameters:

structure

Returns:

8.30.2.4 char∗ add_root (const char ∗ *structure*)

Adds a root to an un-rooted tree in any except bracket notation.

Parameters:

structure

Returns:

8.30.2.5 char∗ expand_Shapiro (const char ∗ *coarse*)

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

Parameters:

coarse

Returns:

8.30.2.6 char∗ expand_Full (const char ∗ *structure*)

Convert the full structure from bracket notation to the expanded notation including root.

Parameters:

structure

Returns:

8.30.2.7 char∗ unexpand_Full (const char ∗ *ffull*)

Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

Parameters:

ffull

Returns:

8.30.2.8 char∗ unweight (const char ∗ *wcoarse*)

Strip weights from any weighted tree.

Parameters:

wcoarse

Returns:

8.30.2.9 void unexpand_aligned_F (char * *align*[2])

Converts two aligned structures in expanded notation.

Takes two aligned structures as produced by tree_edit_distance() function back to bracket notation with '_' as the gap character. The result overwrites the input.

Parameters:

align

8.30.2.10 void parse_structure (const char * *structure*)

Collects a statistic of structure elements of the full structure in bracket notation.

The function writes to the following global variables: loop_size, loop_degree, helix_-size, loops, pairs, unpaired

Parameters:

structure

Returns:

8.31 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/stringdist.h File Reference

Functions for String Alignment.

Include dependency graph for stringdist.h:

Functions

- swString ∗ Make_swString (char ∗string)

  Convert a structure into a format suitable for string_edit_distance().

- float string_edit_distance (swString ∗T1, swString ∗T2)

  Calculate the string edit distance of T1 and T2.

8.31.1 Detailed Description

Functions for String Alignment.

8.31.2 Function Documentation

8.31.2.1 swString∗ Make_swString (char ∗ *string*)

Convert a structure into a format suitable for string_edit_distance().

Parameters:

string

Returns:

8.31.2.2 float string_edit_distance (swString ∗ *T1*, swString ∗ *T2*)

Calculate the string edit distance of T1 and T2.

Parameters:

T1

T2

Returns:

8.32 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/H/subopt.h File Reference

RNAsubopt and density of states declarations.

Include dependency graph for subopt.h:

Functions

- SOLUTION ∗ subopt (char ∗seq, char ∗structure, int delta, FILE ∗fp)

  Returns list of subopt structures or writes to fp.

- SOLUTION ∗ subopt_par (char ∗seq, char ∗structure, paramT ∗parameters, int delta, int is_constrained, int is_circular, FILE ∗fp)

  Returns list of subopt structures or writes to fp.

- SOLUTION ∗ subopt_circ (char ∗seq, char ∗sequence, int delta, FILE ∗fp)

  Returns list of circular subopt structures or writes to fp.

Variables

- int subopt_sorted

  Sort output by energy.

- double print_energy

  printing threshold for use with logML

- int density_of_states [MAXDOS+1]

  The Density of States.

8.32.1 Detailed Description

RNAsubopt and density of states declarations.

8.33 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/treedist.h File Reference

Functions for Tree Edit Distances.

Include dependency graph for treedist.h:

Functions

- Tree ∗ make_tree (char ∗struc)

  Constructs a Tree (essentially the postorder list) of the structure 'struc', for use in tree_edit_distance().

- float tree_edit_distance (Tree ∗T1, Tree ∗T2)

  Calculates the edit distance of the two trees.

- void print_tree (Tree ∗t)

  Print a tree (mainly for debugging).

- void free_tree (Tree ∗t)

  Free the memory allocated for Tree t.

8.33.1 Detailed Description

Functions for Tree Edit Distances.

8.33.2 Function Documentation

8.33.2.1 Tree∗ make_tree (char ∗ *struc*)

Constructs a Tree (essentially the postorder list) of the structure 'struc', for use in tree_edit_distance().

Parameters:

struc may be any rooted structure representation.

Returns:

8.33.2.2 float tree_edit_distance (Tree ∗ *T1*, Tree ∗ *T2*)

Calculates the edit distance of the two trees.

Parameters:

T1

T2

Returns:

8.33.2.3 void free_tree (Tree $*$ *t*)

Free the memory allocated for Tree t.

Parameters:

t

8.34 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/H/utils.h File Reference

Various utility- and helper-functions used throughout the Vienna RNA package.

This graph shows which files directly or indirectly include this file:

Defines

- #define VRNA_INPUT_ERROR 1U
- #define VRNA_INPUT_QUIT 2U
- #define VRNA_INPUT_MISC 4U
- #define VRNA_INPUT_FASTA_HEADER 8U
- #define VRNA_INPUT_SEQUENCE 16U
- #define VRNA_INPUT_CONSTRAINT 32U
- #define VRNA_INPUT_NO_TRUNCATION 256U
- #define VRNA_INPUT_NO_REST 512U
- #define VRNA_INPUT_NO_SPAN 1024U
- #define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U
- #define VRNA_INPUT_BLANK_LINE 4096U
- #define VRNA_INPUT_NOSKIP_COMMENTS 128U
- #define VRNA_INPUT_COMMENT 8192U
- #define VRNA_CONSTRAINT_PIPE 1U
- #define VRNA_CONSTRAINT_DOT 2U
- #define VRNA_CONSTRAINT_X 4U
- #define VRNA_CONSTRAINT_ANG_BRACK 8U
- #define VRNA_CONSTRAINT_RND_BRACK 16U
- #define VRNA_CONSTRAINT_MULTILINE 32U
- #define VRNA_CONSTRAINT_NO_HEADER 64U
- #define VRNA_CONSTRAINT_ALL 128U
- #define VRNA_CONSTRAINT_G 256U
- #define VRNA_OPTION_MULTILINE 32U
- #define MIN2(A, B) ((A) < (B) ? (A) : (B))
- #define MAX2(A, B) ((A) > (B) ? (A) : (B))
- #define MIN3(A, B, C) (MIN2((MIN2((A),(B))) ,(C)))
- #define MAX3(A, B, C) (MAX2((MAX2((A),(B))) ,(C)))
- #define XSTR(s) STR(s)
- #define STR(s) #s
- #define FILENAME_MAX_LENGTH 80

  Maximum length of filenames that are generated by our programs.

- #define FILENAME_ID_LENGTH 42

  Maximum length of id taken from fasta header for filename generation.

Functions

- void ∗ space (unsigned size)

  Allocate space safely.

- void ∗ xrealloc (void ∗p, unsigned size)

  Reallocate space safely.

- void nrerror (const char message[ ])

  Die with an error message.

- void warn_user (const char message[ ])

  Print a warning message.

- void init_rand (void)

  Make random number seeds.

- double urn (void)

  get a random number from [0..1]

- int int_urn (int from, int to)

  Generates a pseudo random integer in a specified range.

- char ∗ time_stamp (void)

  Get a timestamp.

- char ∗ random_string (int l, const char symbols[ ])

  Create a random string using characters from a specified symbol set.

- int hamming (const char ∗s1, const char ∗s2)

  Calculate hamming distance between two sequences.

- int hamming_bound (const char ∗s1, const char ∗s2, int n)

  Calculate hamming distance between two sequences up to a specified length.

- char ∗ get_line (FILE ∗fp)

  Read a line of arbitrary length from a stream.

- unsigned int get_input_line (char ∗∗string, unsigned int options)
- unsigned int read_record (char ∗∗header, char ∗∗sequence, char ∗∗∗rest, unsigned int options)

  Get a data record from stdin.

- char ∗ pack_structure (const char ∗struc)

  Pack secondary secondary structure, 5:1 compression using base 3 encoding.

- char ∗ unpack_structure (const char ∗packed)

  Unpack secondary structure previously packed with pack_structure().

- short ∗ make_pair_table (const char ∗structure)

  Create a pair table of a secondary structure.

- short ∗ copy_pair_table (const short ∗pt)

  Get an exact copy of a pair table.

- short ∗ alimake_pair_table (const char ∗structure)
- short ∗ make_pair_table_snoop (const char ∗structure)
- int ∗ make_loop_index_pt (short ∗pt)

  Compute the "base pair" distance between two secondary structures s1 and s2.

- void print_tty_input_seq (void)

  Print a line to stdout *that asks for an input sequence.*

- void print_tty_input_seq_str (const char ∗s)

  Print a line with a user defined string and a ruler to stdout.

- void print_tty_constraint_full (void)

  Print structure constraint characters to stdout (full constraint support).

- void print_tty_constraint (unsigned int option)

  Print structure constraint characters to stdout. (constraint support is specified by option parameter).

- void str_DNA2RNA (char ∗sequence)

  Convert a DNA input sequence to RNA alphabet.

- void str_uppercase (char ∗sequence)

  Convert an input sequence to uppercase.

- int ∗ get_iindx (unsigned int length)

  Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.

- int ∗ get_indx (unsigned int length)

  Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.

- void constrain_ptypes (const char ∗constraint, unsigned int length, char ∗ptype, int ∗BP, int min_loop_size, unsigned int idx_type)

  Insert constraining pair types according to constraint structure string.

Variables

- unsigned short xsubi [3]

    Current 48 bit random number.

8.34.1 Detailed Description

Various utility- and helper-functions used throughout the Vienna RNA package.

8.34.2 Define Documentation

8.34.2.1 #define VRNA_INPUT_ERROR 1U

Output flag of get_input_line(): "An ERROR has occured, maybe EOF"

8.34.2.2 #define VRNA_INPUT_QUIT 2U

Output flag of get_input_line(): "the user requested quitting the program"

8.34.2.3 #define VRNA_INPUT_MISC 4U

Output flag of get_input_line(): "something was read"

8.34.2.4 #define VRNA_INPUT_FASTA_HEADER 8U

Input/Output flag of get_input_line():

if used as input option this tells get_input_line() that the data to be read should comply with the FASTA format

the function will return this flag if a fasta header was read

8.34.2.5 #define VRNA_INPUT_SEQUENCE 16U

Input flag for get_input_line():

Tell get_input_line() that we assume to read a nucleotide sequence

8.34.2.6 #define VRNA_INPUT_CONSTRAINT 32U

Input flag for get_input_line():

Tell get_input_line() that we assume to read a structure constraint

8.34.2.7 #define VRNA_INPUT_NO_TRUNCATION 256U

Input switch for get_input_line(): "do not trunkate the line by eliminating white spaces at end of line"

8.34.2.8 #define VRNA_INPUT_NO_REST 512U

Input switch for read_record(): "do fill rest array"

8.34.2.9 #define VRNA_INPUT_NO_SPAN 1024U

Input switch for read_record(): "never allow data to span more than one line"

8.34.2.10 #define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U

Input switch for read_record(): "do not skip empty lines"

8.34.2.11 #define VRNA_INPUT_BLANK_LINE 4096U

Output flag for read_record(): "read an empty line"

8.34.2.12 #define VRNA_INPUT_NOSKIP_COMMENTS 128U

Input switch for get_input_line(): "do not skip comment lines"

8.34.2.13 #define VRNA_INPUT_COMMENT 8192U

Output flag for read_record(): "read a comment"

8.34.2.14 #define VRNA_CONSTRAINT_PIPE 1U

pipe sign '|' switch for structure constraints (paired with another base)

8.34.2.15 #define VRNA_CONSTRAINT_DOT 2U

dot '.' switch for structure constraints (no constraint at all)

8.34.2.16 #define VRNA_CONSTRAINT_X 4U

'x' switch for structure constraint (base must not pair)

8.34.2.17 #define VRNA_CONSTRAINT_ANG_BRACK 8U

angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)

8.34.2.18 #define VRNA_CONSTRAINT_RND_BRACK 16U

round brackets '(',')' switch for structure constraint (base i pairs base j)

8.34.2.19 #define VRNA_CONSTRAINT_MULTILINE 32U

constraint may span over several lines

8.34.2.20 #define VRNA_CONSTRAINT_NO_HEADER 64U

do not print the header information line

8.34.2.21 #define VRNA_CONSTRAINT_ALL 128U

placeholder for all constraining characters

8.34.2.22 #define VRNA_CONSTRAINT_G 256U

'+' switch for structure constraint (base is involved in a gquad)

8.34.2.23 #define VRNA_OPTION_MULTILINE 32U

Tell a function that an input is assumed to span several lines if used as input-option A function might also be returning this state telling that it has read data from multiple lines.

See also:

extract_record_rest_structure(), read_record(), getConstraint()

8.34.2.24 #define MIN2(A, B) ((A) < (B) ? (A) : (B))

Get the minimum of two comparable values

8.34.2.25 #define MAX2(A, B) ((A) > (B) ? (A) : (B))

Get the maximum of two comparable values

8.34.2.26 #define MIN3(A, B, C) (MIN2((MIN2((A),(B))) ,(C)))

Get the minimum of three comparable values

8.34.2.27 #define MAX3(A, B, C) (MAX2((MAX2((A),(B))) ,(C)))

Get the maximum of three comparable values

8.34.2.28 #define XSTR(s) STR(s)

Stringify a macro after expansion

8.34.2.29 #define STR(s) #s

Stringify a macro argument

8.34.2.30 #define FILENAME_MAX_LENGTH 80

Maximum length of filenames that are generated by our programs.

This definition should be used throughout the complete ViennaRNA package wherever a static array holding filenames of output files is declared.

8.34.2.31 #define FILENAME_ID_LENGTH 42

Maximum length of id taken from fasta header for filename generation.

this has to be smaller than FILENAME_MAX_LENGTH since in most cases, some suffix will be appended to the ID

8.34.3 Function Documentation

8.34.3.1 void∗ space (unsigned *size*)

Allocate space safely.

Parameters:

size The size of the memory to be allocated in bytes

Returns:

A pointer to the allocated memory

8.34.3.2 void* xrealloc (void * *p*, unsigned *size*)

Reallocate space safely.

Parameters:

p A pointer to the memory region to be reallocated

size The size of the memory to be allocated in bytes

Returns:

A pointer to the newly allocated memory

8.34.3.3 void nrerror (const char *message*[ ])

Die with an error message.

See also:

warn_user()

Parameters:

message The error message to be printed before exiting with ’FAILURE’

8.34.3.4 void warn_user (const char *message*[ ])

Print a warning message.

Print a warning message to *stderr*

Parameters:

message The warning message

8.34.3.5 double urn (void)

get a random number from [0..1]

Note:

Usually implemented by calling *erand48()*.

Returns:

A random number in range [0..1]

8.34.3.6 int int_urn (int *from*, int *to*)

Generates a pseudo random integer in a specified range.

Parameters:

from The first number in range
to The last number in range

Returns:

A pseudo random number in range [from, to]

8.34.3.7 char∗ time_stamp (void)

Get a timestamp.

Returns a string containing the current date in the format

```
Fri Mar 19 21:10:57 1993
```

Returns:

A string containing the timestamp

8.34.3.8 char∗ random_string (int *l*, const char *symbols*[ ])

Create a random string using characters from a specified symbol set.

Parameters:

l The length of the sequence
symbols The symbol set

Returns:

A random string of length 'l' containing characters from the symbolset

8.34.3.9 int hamming (const char ∗ *s1*, const char ∗ *s2*)

Calculate hamming distance between two sequences.

Calculate the number of positions in which

Parameters:

s1 The first sequence
s2 The second sequence

Returns:

The hamming distance between s1 and s2

8.34.3.10 int hamming_bound (const char ∗ *s1*, const char ∗ *s2*, int *n*)

Calculate hamming distance between two sequences up to a specified length.

This function is similar to hamming() but instead of comparing both sequences up to their actual length only the first 'n' characters are taken into account

Parameters:

s1 The first sequence

s2 The second sequence

Returns:

The hamming distance between s1 and s2

8.34.3.11 char∗ get_line (FILE ∗ *fp*)

Read a line of arbitrary length from a stream.

Returns a pointer to the resulting string. The necessary memory is allocated and should be released using *free()* when the string is no longer needed.

Parameters:

fp A file pointer to the stream where the function should read from

Returns:

A pointer to the resulting string

8.34.3.12 unsigned int get_input_line (char ∗∗ *string*, unsigned int *options*)

Retrieve a line from 'stdin' savely while skipping comment characters and other features This function returns the type of input it has read if recognized. An option argument allows to switch between different reading modes.

Currently available options are:

VRNA_INPUT_NOPRINT_COMMENTS, VRNA_INPUT_NOSKIP_COMMENTS, VRNA_INPUT_NOELIM_WS_SUFFIX

pass a collection of options as one value like this:

```
get_input_line(string, option_1 | option_2 | option_n)
```

If the function recognizes the type of input, it will report it in the return value. It also reports if a user defined 'quit' command (@-sign on 'stdin') was given. Possible return values are:

VRNA_INPUT_FASTA_HEADER, VRNA_INPUT_ERROR, VRNA_INPUT_MISC, VRNA_INPUT_QUIT

Parameters:

string A pointer to the character array that contains the line read

options A collection of options for switching the functions behavior

Returns:

A flag with information about what has been read

8.34.3.13 unsigned int read_record (char ∗∗ *header*, char ∗∗ *sequence*, char ∗∗∗ *rest*, unsigned int *options*)

Get a data record from stdin.

This function may be used to obtain complete datasets from stdin. A dataset is always defined to contain at least a sequence. If data on stdin starts with a fasta header, i.e. a line like

```
>some header info
```

then read_record() will assume that the sequence that follows the header may span over several lines. To disable this behavior and to assign a single line to the argument 'sequence' one can pass VRNA_INPUT_NO_SPAN in the 'options' argument. If no fasta header is read in the beginning of a data block, a sequence must not span over multiple lines!

Unless the options VRNA_INPUT_NOSKIP_COMMENTS or VRNA_INPUT_-NOSKIP_BLANK_LINES are passed, a sequence may be interrupted by lines starting with a comment character or empty lines.

A sequence is regarded as completely read if it was either assumed to not span over multiple lines, a secondary structure or structure constraint follows the sequence on the next line or a new header marks the beginning of a new sequence...

All lines following the sequence (this includes comments) and not initiating a new dataset are available through the line-array 'rest'. Here one can usually find the structure constraint or other information belonging to the current dataset. Filling of 'rest' may be prevented by passing VRNA_INPUT_NO_REST to the options argument.

Note:

This function will exit any program with an error message if no sequence could be read!

The main purpose of this function is to be able to easily parse blocks of data from stdin in the header of a loop where all calculations for the appropriate data is done inside the loop. The loop may be then left on certain return values, e.g.:

```
char *id, *seq, **rest;
int  i;
while(!(read_record(&id, &seq, &rest, 0) & (VRNA_INPUT_ERROR | VRNA_INPUT_QUIT))){
  if(id) printf("%s\n", id);
  printf("%s\n", seq);
  if(rest)
    for(i=0;rest[i];i++)
      printf("%s\n", rest[i]);
}
```

In the example above, the while loop will be terminated when read_record() returns either an error or a user initiated quit request.

As long as data is read from stdin, the id is printed if it is available for the current block of data. The sequence will be printed in any case and if some more lines belong to the current block of data each line will be printed as well.

Note:

Do not forget to free the memory occupied by header, sequence and rest!

Parameters:

header A pointer which will be set such that it points to the header of the record

sequence A pointer which will be set such that it points to the sequence of the record

rest A pointer which will be set such that it points to an array of lines which also belong to the record

options Some options which may be passed to alter the behavior of the function, use 0 for no options

Returns:

A flag with information about what the function actually did read

8.34.3.14 char∗ pack_structure (const char ∗ *struc*)

Pack secondary secondary structure, 5:1 compression using base 3 encoding.

Returns a binary string encoding of the secondary structure using a 5:1 compression scheme. The string is NULL terminated and can therefore be used with standard string functions such as strcmp(). Useful for programs that need to keep many structures in memory.

Parameters:

struc The secondary structure in dot-bracket notation

Returns:

The binary encoded structure

8.34.3.15 char∗ unpack_structure (const char ∗ *packed*)

Unpack secondary structure previously packed with pack_structure().

Translate a compressed binary string produced by pack_structure() back into the familiar dot-bracket notation.

Parameters:

packed The binary encoded packed secondary structure

Returns:

The unpacked secondary structure in dot-bracket notation

8.34.3.16 short∗ make_pair_table (const char ∗ *structure*)

Create a pair table of a secondary structure.

Returns a newly allocated table, such that table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

Parameters:

structure The secondary structure in dot-bracket notation

Returns:

A pointer to the created pair_table

8.34.3.17 short∗ copy_pair_table (const short ∗ *pt*)

Get an exact copy of a pair table.

Parameters:

pt The pair table to be copied

Returns:

A pointer to the copy of 'pt'

8.34.3.18 short∗ alimake_pair_table (const char ∗ *structure*)

Pair table for snoop align

8.34.3.19 short∗ make_pair_table_snoop (const char ∗ *structure*)

returns a newly allocated table, such that: table[i]=j if (i.j) pair or 0 if i is unpaired, table[0] contains the length of the structure. The special pseudoknotted H/ACA-mRNA structure is taken into account.

8.34.3.20 int∗ make_loop_index_pt (short ∗ *pt*)

Compute the "base pair" distance between two secondary structures s1 and s2.

The sequences should have the same length. dist = number of base pairs in one structure but not in the other same as edit distance with open-pair close-pair as move-set

Parameters:

str1 First structure in dot-bracket notation

str2 Second structure in dot-bracket notation

Returns:

The base pair distance between str1 and str2

8.34.3.21 void print_tty_input_seq (void)

Print a line to *stdout* that asks for an input sequence.

There will also be a ruler (scale line) printed that helps orientation of the sequence positions

8.34.3.22 void print_tty_input_seq_str (const char ∗ *s*)

Print a line with a user defined string and a ruler to stdout.

(usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

Parameters:

s A user defined string that will be printed to stdout

8.34.3.23 void print_tty_constraint (unsigned int *option*)

Print structure constraint characters to stdout. (constraint support is specified by option parameter).

Currently available options are:

VRNA_CONSTRAINT_PIPE (paired with another base)

VRNA_CONSTRAINT_DOT (no constraint at all)

VRNA_CONSTRAINT_X (base must not pair)

VRNA_CONSTRAINT_ANG_BRACK (paired downstream/upstream)

VRNA_CONSTRAINT_RND_BRACK (base i pairs base j)

pass a collection of options as one value like this:

```
print_tty_constraint(option_1 | option_2 | option_n)
```

Parameters:

option Option switch that tells which constraint help will be printed

8.34.3.24 void str_DNA2RNA (char * *sequence*)

Convert a DNA input sequence to RNA alphabet.

This function substitudes *T* and *t* with *U* and *u*, respectively

Parameters:

sequence The sequence to be converted

8.34.3.25 void str_uppercase (char * *sequence*)

Convert an input sequence to uppercase.

Parameters:

sequence The sequence to be converted

8.34.3.26 int* get_iindx (unsigned int *length*)

Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.

Access of a position "(i,j)" is then accomplished by using

```
(i,j) ~ iindx[i]-j
```

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNAPackage

Consult the implemented code to find out about the mapping formula ;)

See also:

get_indx()

Parameters:

length The length of the RNA sequence

Returns:

The mapper array

8.34.3.27 int∗ get_indx (unsigned int *length*)

Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.

Access of a position "(i,j)" is then accomplished by using

```
(i,j) ~ indx[j]+i
```

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNAPackage

Consult the implemented code to find out about the mapping formula ;)

See also:

get_iindx()

Parameters:

length The length of the RNA sequence

Returns:

The mapper array

8.34.3.28 void constrain_ptypes (const char ∗ *constraint*, unsigned int *length*, char ∗ *ptype*, int ∗ *BP*, int *min_loop_size*, unsigned int *idx_type*)

Insert constraining pair types according to constraint structure string.

See also:

get_indx(), get_iindx()

Parameters:

constraint The structure constraint string

length The actual length of the sequence (constraint may be shorter)

ptype A pointer to the basepair type array

min_loop_size The minimal loop size (usually TURN)

idx_type Define the access type for base pair type array (0 = indx, 1 = iindx)

8.34.4 Variable Documentation

8.34.4.1 unsigned short xsubi[3]

Current 48 bit random number.

This variable is used by urn(). These should be set to some random number seeds before the first call to urn().

See also:

urn()

8.35 /scratch/2/miladim/sparse/installRep/ViennaRNA-2.1.2/lib/1.8.4_epars.h File Reference

Free energy parameters for parameter file conversion.

8.35.1 Detailed Description

Free energy parameters for parameter file conversion.

This file contains the free energy parameters used in ViennaRNAPackage 1.8.4. They are summarized in:

D.H.Mathews, J. Sabina, M. ZUker, D.H. Turner "Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure" JMB, 288, pp 911-940, 1999

Enthalpies taken from:

A. Walter, D Turner, J Kim, M Lyttle, P M"uller, D Mathews, M Zuker "Coaxial stckaing of helices enhances binding of oligoribonucleotides.." PNAS, 91, pp 9218-9222, 1994

D.H. Turner, N. Sugimoto, and S.M. Freier. "RNA Structure Prediction", Ann. Rev. Biophys. Biophys. Chem. 17, 167-192, 1988.

John A.Jaeger, Douglas H.Turner, and Michael Zuker. "Improved predictions of secondary structures for RNA", PNAS, 86, 7706-7710, October 1989.

L. He, R. Kierzek, J. SantaLucia, A.E. Walter, D.H. Turner "Nearest-Neughbor Parameters for GU Mismatches...." Biochemistry 1991, 30 11124-11132

A.E. Peritz, R. Kierzek, N, Sugimoto, D.H. Turner "Thermodynamic Study of Internal Loops in Oligoribonucleotides..." Biochemistry 1991, 30, 6428–6435

8.36 /scratch/2/miladim/sparse/installRep/Vienna-RNA-2.1.2/lib/1.8.4_intloops.h File Reference

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

8.36.1 Detailed Description

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

Chapter 9

RNAlib-2.1.2 Page Documentation

9.1 Parsing and Comparing - Functions to Manipulate Structures

Representations of Secondary Structures

The standard representation of a secondary structure is the *bracket notation*, where matching brackets symbolize base pairs and unpaired bases are shown as dots. Alternatively, one may use two types of node labels, 'P' for paired and 'U' for unpaired; a dot is then replaced by '(U)', and each closed bracket is assigned an additional identifier 'P'. We call this the expanded notation. In fontana:1993b a condensed representation of the secondary structure is proposed, the so-called homeomorphically irreducible tree (HIT) representation. Here a stack is represented as a single pair of matching brackets labeled 'P' and weighted by the number of base pairs. Correspondingly, a contiguous strain of unpaired bases is shown as one pair of matching brackets labeled 'U' and weighted by its length. Generally any string consisting of matching brackets and identifiers is equivalent to a plane tree with as many different types of nodes as there are identifiers.

Bruce Shapiro proposed a coarse grained representation shapiro:1988, which, does not retain the full information of the secondary structure. He represents the different structure elements by single matching brackets and labels them as 'H' (hairpin loop), 'I' (interior loop), 'B' (bulge), 'M' (multi-loop), and 'S' (stack). We extend his alphabet by an extra letter for external elements 'E'. Again these identifiers may be followed by a weight corresponding to the number of unpaired bases or base pairs in the structure element. All tree representations (except for the dot-bracket form) can be encapsulated into a virtual root (labeled 'R'), see the example below.

The following example illustrates the different linear tree representations used by the package. All lines show the same secondary structure.

```
a) .((((..(((...)))..((..)))).)).
```

```
   (U)((((U)(U)((((U)(U)(U)P)P)P)(U)(U)(((U)(U)P)P)P)P)(U)P)P)(U)
b) (U)(((U2)((U3)P3)(U2)((U2)P2)P2)(U)P2)(U)
c) (((H)(H)M)B)
   ((((((H)S)((H)S)M)S)B)S)
   (((((((H)S)((H)S)M)S)B)S)E)
d) (((((((((H3)S3)((H2)S2)M4)S2)B1)S2)E2)R)
```

Above: Tree representations of secondary structures. a) Full structure: the first line shows the more convenient condensed notation which is used by our programs; the second line shows the rather clumsy expanded notation for completeness, b) HIT structure, c) different versions of coarse grained structures: the second line is exactly Shapiro's representation, the first line is obtained by neglecting the stems. Since each loop is closed by a unique stem, these two lines are equivalent. The third line is an extension taking into account also the external digits. d) weighted coarse structure, this time including the virtual root.

For the output of aligned structures from string editing, different representations are needed, where we put the label on both sides. The above examples for tree representations would then look like:

```
a) (UU)(P(P(P(P(UU)(UU)(P(P(P(UU)(UU)(UU)P)P)P)(UU)(UU)(P(P(UU)(U...
b) (UU)(P2(P2(U2U2)(P2(U3U3)P3)(U2U2)(P2(U2U2)P2)P2)(UU)P2)(UU)
c) (B(M(HH)(HH)M)B)
   (S(B(S(M(S(HH)S)(S(HH)S)M)S)B)S)
   (E(S(B(S(M(S(HH)S)(S(HH)S)M)S)B)S)E)
d) (R(E2(S2(B1(S2(M4(S3(H3)S3)((H2)S2)M4)S2)B1)S2)E2)R)
```

Aligned structures additionally contain the gap character '_'.

Parsing and Coarse Graining of Structures

Several functions are provided for parsing structures and converting to different representations.

```
char  *expand_Full(const char *structure)
```

expand_Full()

```
char *b2HIT (const char *structure)
```

b2HIT()

```
char *b2C (const char *structure)
```

b2C()

```
char *b2Shapiro (const char *structure)
```

b2Shapiro()

```
char  *expand_Shapiro (const char *coarse);
```

expand_Shapiro()

```
char *add_root (const char *structure)
```

add_root()

```
char  *unexpand_Full (const char *ffull)
```

unexpand_Full()

```
char  *unweight (const char *wcoarse)
```

unweight()

```
void   unexpand_aligned_F (char *align[2])
```

unexpand_aligned_F()

```
void   parse_structure (const char *structure)
```

parse_structure()

See also:

RNAstruct.h for prototypes and more detailed description

Distance Measures

A simple measure of dissimilarity between secondary structures of equal length is the base pair distance, given by the number of pairs present in only one of the two structures being compared. I.e. the number of base pairs that have to be opened or closed to transform one structure into the other. It is therefore particularly useful for comparing structures on the same sequence. It is implemented by

```
int bp_distance(const char *str1,
                const char *str2)
```

bp_distance()

For other cases a distance measure that allows for gaps is preferable. We can define distances between structures as edit distances between trees or their string representations. In the case of string distances this is the same as "sequence alignment". Given a set of edit operations and edit costs, the edit distance is given by the minimum sum of the costs along an edit path converting one object into the other. Edit distances like these always define a metric. The edit operations used by us are insertion, deletion and replacement of nodes. String editing does not pay attention to the matching of brackets, while in tree editing matching brackets represent a single node of the tree. Tree editing is therefore usually preferable, although somewhat slower. String edit distances are always smaller or equal to tree edit distances.

The different level of detail in the structure representations defined above naturally leads to different measures of distance. For full structures we use a cost of 1 for deletion or insertion of an unpaired base and 2 for a base pair. Replacing an unpaired base for a pair incurs a cost of 1.

Two cost matrices are provided for coarse grained structures:

```
/*  Null,   H,    B,    I,    M,    S,    E    */
   {   0,    2,    2,    2,    2,    1,    1},   /* Null replaced */
   {   2,    0,    2,    2,    2, INF,  INF},   /* H    replaced */
   {   2,    2,    0,    1,    2, INF,  INF},   /* B    replaced */
   {   2,    2,    1,    0,    2, INF,  INF},   /* I    replaced */
   {   2,    2,    2,    2,    0, INF,  INF},   /* M    replaced */
   {   1, INF,  INF,  INF,  INF,    0,  INF},   /* S    replaced */
   {   1, INF,  INF,  INF,  INF,  INF,    0},   /* E    replaced */


/* Null,   H,    B,    I,    M,    S,    E   */
   {   0, 100,    5,    5,   75,    5,    5},   /* Null replaced */
   { 100,    0,    8,    8,    8, INF,  INF},   /* H    replaced */
   {   5,    8,    0,    3,    8, INF,  INF},   /* B    replaced */
   {   5,    8,    3,    0,    8, INF,  INF},   /* I    replaced */
   {  75,    8,    8,    8,    0, INF,  INF},   /* M    replaced */
   {   5, INF,  INF,  INF,  INF,    0,  INF},   /* S    replaced */
   {   5, INF,  INF,  INF,  INF,  INF,    0},   /* E    replaced */
```

The lower matrix uses the costs given in shapiro:1990. All distance functions use the following global variables:

```
int  cost_matrix;
```

cost_matrix

```
int   edit_backtrack;
```

edit_backtrack

```
char *aligned_line[4];
```

aligned_line

See also:

utils.h, dist_vars.h and stringdist.h for more details

Functions for Tree Edit Distances

```
Tree   *make_tree (char *struc)
```

make_tree()

```
float   tree_edit_distance (Tree *T1,
                            Tree *T2)
```

tree_edit_distance()

```
void    free_tree(Tree *t)
```

free_tree()

See also:

dist_vars.h and treedist.h for prototypes and more detailed descriptions

Functions for String Alignment

```
swString *Make_swString (char *string)
```

Make_swString()

```
float     string_edit_distance (swString *T1,
                                swString *T2)
```

string_edit_distance()

See also:

dist_vars.h and stringdist.h for prototypes and more detailed descriptions

Functions for Comparison of Base Pair Probabilities For comparison of base pair probability matrices, the matrices are first condensed into probability profiles which are the compared by alignment.

```
float *Make_bp_profile_bppm ( double *bppm,
                              int length)
```

Make_bp_profile_bppm()

```
float profile_edit_distance ( const float *T1,
                              const float *T2)
```

profile_edit_distance()

See also:

ProfileDist.h for prototypes and more details of the above functions

Next Page: Utilities

9.2 Example - A Small Example Program

_utils Utilities - Odds and Ends

Table of Contents

9.2.1 Producing secondary structure graphs

```
int PS_rna_plot ( char *string,
                  char *structure,
                  char *file)
```

PS_rna_plot()

```
int PS_rna_plot_a (
            char *string,
            char *structure,
            char *file,
            char *pre,
            char *post)
```

PS_rna_plot_a()

```
int gmlRNA (char *string,
            char *structure,
            char *ssfile,
            char option)
```

gmlRNA()

```
int ssv_rna_plot (char *string,
                  char *structure,
                  char *ssfile)
```

ssv_rna_plot()

```
int svg_rna_plot (char *string,
                  char *structure,
                  char *ssfile)
```

svg_rna_plot()

```
int xrna_plot ( char *string,
                char *structure,
                char *ssfile)
```

xrna_plot()

```
int rna_plot_type
```

rna_plot_type

Two low-level functions provide direct access to the graph lauyouting algorithms:

```
int simple_xy_coordinates ( short *pair_table,
                            float *X,
                            float *Y)
```

simple_xy_coordinates()

```
int naview_xy_coordinates ( short *pair_table,
                            float *X,
                            float *Y)
```

naview_xy_coordinates()

See also:

PS_dot.h and naview.h for more detailed descriptions.

9.2.2 Producing (colored) dot plots for base pair probabilities

```
int PS_color_dot_plot ( char *string,
                        cpair *pi,
                        char *filename)
```

PS_color_dot_plot()

```
int PS_color_dot_plot_turn (char *seq,
                            cpair *pi,
                            char *filename,
                            int winSize)
```

PS_color_dot_plot_turn()

```
int PS_dot_plot_list (char *seq,
                      char *filename,
                      plist *pl,
                      plist *mf,
                      char *comment)
```

PS_dot_plot_list()

```
int PS_dot_plot_turn (char *seq,
                      struct plist *pl,
                      char *filename,
                      int winSize)
```

PS_dot_plot_turn()

See also:

PS_dot.h for more detailed descriptions.

9.2.3 Producing (colored) alignments

```
int PS_color_aln (
            const char *structure,
            const char *filename,
            const char *seqs[],
            const char *names[])
```

PS_color_aln()

9.2.4 RNA sequence related utilities

Several functions provide useful applications to RNA sequences

```
char  *random_string (int l,
                      const char symbols[])
```

random_string()

```
int   hamming ( const char *s1,
                const char *s2)
```

hamming()

```
void str_DNA2RNA(char *sequence);
```

str_DNA2RNA()

```
void str_uppercase(char *sequence);
```

str_uppercase()

9.2.5 RNA secondary structure related utilities

```
char *pack_structure (const char *struc)
```

pack_structure()

```
char *unpack_structure (const char *packed)
```

unpack_structure()

```
short *make_pair_table (const char *structure)
```

make_pair_table()

```
short *copy_pair_table (const short *pt)
```

copy_pair_table()

9.2.6 Miscellaneous Utilities

```
void print_tty_input_seq (void)
```

print_tty_input_seq()

```
void print_tty_constraint_full (void)
```

print_tty_constraint_full()

```
void print_tty_constraint (unsigned int option)
```

print_tty_constraint()

```
int   *get_iindx (unsigned int length)
```

get_iindx()

```
int   *get_indx (unsigned int length)
```

get_indx()

```
void constrain_ptypes (
                const char *constraint,
                unsigned int length,
                char *ptype,
                int *BP,
                int min_loop_size,
                unsigned int idx_type)
```

constrain_ptypes()

```
char  *get_line(FILE *fp);
```

get_line()

```
unsigned int read_record(
                char **header,
                char **sequence,
                char ***rest,
                unsigned int options);
```

read_record()

```
char  *time_stamp (void)
```

time_stamp()

```
void warn_user (const char message[])
```

warn_user()

```
void nrerror (const char message[])
```

nrerror()

```
void   init_rand (void)
```

init_rand()

```
unsigned short xsubi[3];
```

xsubi

```
double urn (void)
```

urn()

```
int    int_urn (int from, int to)
```

int_urn()

```
void  *space (unsigned size)
```

space()

```
void  *xrealloc ( void *p,
                  unsigned size)
```

xrealloc()

See also:

utils.h for a complete overview and detailed description of the utility functions

Next Page: Examples

The following program exercises most commonly used functions of the library. The program folds two sequences using both the mfe and partition function algorithms and calculates the tree edit and profile distance of the resulting structures and base pairing probabilities.

```
 {.c}
#include  <stdio.h>
#include  <stdlib.h>
#include  <math.h>
#include  <string.h>
#include  "utils.h"
#include  "fold_vars.h"
#include  "fold.h"
#include  "part_func.h"
#include  "inverse.h"
#include  "RNAstruct.h"
#include  "treedist.h"
#include  "stringdist.h"
#include  "profiledist.h"

void main()
{
   char *seq1="CGCAGGGAUACCCGCG", *seq2="GCGCCCAUAGGGACGC",
        *struct1,* struct2,* xstruc;
   float e1, e2, tree_dist, string_dist, profile_dist, kT;
   Tree *T1, *T2;
   swString *S1, *S2;
   float *pf1, *pf2;
   FLT_OR_DBL *bppm;
   /* fold at 30C instead of the default 37C */
   temperature = 30.;      /* must be set *before* initializing  */

   /* allocate memory for structure and fold */
   struct1 = (char* ) space(sizeof(char)*(strlen(seq1)+1));
   e1 =  fold(seq1, struct1);

   struct2 = (char* ) space(sizeof(char)*(strlen(seq2)+1));
   e2 =  fold(seq2, struct2);

   free_arrays();     /* free arrays used in fold() */

   /* produce tree and string representations for comparison */
   xstruc = expand_Full(struct1);
   T1 = make_tree(xstruc);
   S1 = Make_swString(xstruc);
   free(xstruc);

   xstruc = expand_Full(struct2);
   T2 = make_tree(xstruc);
   S2 = Make_swString(xstruc);
   free(xstruc);

   /* calculate tree edit distance and aligned structures with gaps */
   edit_backtrack = 1;
   tree_dist = tree_edit_distance(T1, T2);
   free_tree(T1); free_tree(T2);
   unexpand_aligned_F(aligned_line);
   printf("%s\n%s  %3.2f\n", aligned_line[0], aligned_line[1], tree_dist);

   /* same thing using string edit (alignment) distance */
   string_dist = string_edit_distance(S1, S2);
   free(S1); free(S2);
   printf("%s  mfe=%5.2f\n%s  mfe=%5.2f  dist=%3.2f\n",
          aligned_line[0], e1, aligned_line[1], e2, string_dist);

   /* for longer sequences one should also set a scaling factor for
      partition function folding, e.g: */
   kT = (temperature+273.15)*1.98717/1000.;  /* kT in kcal/mol */
   pf_scale = exp(-e1/kT/strlen(seq1));

   /* calculate partition function and base pair probabilities */
   e1 = pf_fold(seq1, struct1);
   /* get the base pair probability matrix for the previous run of pf_fold() */
   bppm = export_bppm();
   pf1 = Make_bp_profile_bppm(bppm, strlen(seq1));

   e2 = pf_fold(seq2, struct2);
   /* get the base pair probability matrix for the previous run of pf_fold() */
   bppm = export_bppm();
   pf2 = Make_bp_profile_bppm(bppm, strlen(seq2));

   free_pf_arrays();  /* free space allocated for pf_fold() */

   profile_dist = profile_edit_distance(pf1, pf2);
   printf("%s  free energy=%5.2f\n%s  free energy=%5.2f  dist=%3.2f\n",
          aligned_line[0], e1, aligned_line[1], e2, profile_dist);

   free_profile(pf1); free_profile(pf2);
}
```

In a typical Unix environment you would compile this program using:

```
cc ${OPENMP_CFLAGS} -c example.c -I${hpath}
```

and link using

```
cc ${OPENMP_CFLAGS} -o example -L${lpath} -lRNA -lm
```

where *${hpath}* and *${lpath}* point to the location of the header files and library, respectively.

Note:

As default, the RNAlib is compiled with build-in *OpenMP* multithreading support. Thus, when linking your own object files to the library you have to pass the compiler specific *${OPENMP_CFLAGS}* (e.g. '-fopenmp' for **gcc**) even if your code does not use openmp specific code. However, in that case the *OpenMP* flags may be ommited when compiling example.c

9.3 Deprecated List

Global initialize_cofold {This function is obsolete and will be removed soon!}

Global LoopEnergy {This function is deprecated and will be removed soon. Use E_IntLoop() instead!}

Global HairpinE {This function is deprecated and will be removed soon. Use E_-Hairpin() instead!}

Global initialize_fold {This function is deprecated and will be removed soon!}

Global energy_of_struct This function is deprecated and should not be used in future programs! Use energy_of_structure() instead!

Global energy_of_struct_pt This function is deprecated and should not be used in future programs! Use energy_of_structure_pt() instead!

Global energy_of_circ_struct This function is deprecated and should not be used in future programs Use energy_of_circ_structure() instead!

Global base_pair Do not use this variable anymore!

Global pr Do not use this variable anymore!

Global iindx Do not use this variable anymore!

Global init_pf_fold This function is obsolete and will be removed soon!

Global centroid This function is deprecated and should not be used anymore as it is not threadsafe!

Global mean_bp_dist This function is not threadsafe and should not be used anymore. Use mean_bp_distance() instead!

Global expLoopEnergy Use exp_E_IntLoop() from loop_energies.h instead

Global expHairpinEnergy Use exp_E_Hairpin() from loop_energies.h instead

Global get_plist { This function is deprecated and will be removed soon!} use assign_plist_from_pr() instead!

Global init_co_pf_fold { This function is deprecated and will be removed soon!}

Global Make_bp_profile This function is deprecated and will be removed soon! See Make_bp_profile_bppm() for a replacement

Global PS_dot_plot This function is deprecated and will be removed soon! Use PS_-dot_plot_list() instead!

Index