# RNAlib-2.4.6

# Contents

# Chapter 1

# Main Page

## 1.1 A Library for predicting and comparing RNA secondary structures

The core of the ViennaRNA Package ([10], [8]) 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/RNA

**Date**

  1994-2018

**Authors**

  Ivo Hofacker, Peter Stadler, Ronny Lorenz, and so many more

## 1.2 License

## 1.3 Contributors

Over the past decades since the `ViennaRNA Package` first sprang to life as part of Ivo Hofackers PhD project, several different authors contributed more and more algorithm implementations. In 2008, Ronny Lorenz took over the extensive task to harmonize and simplify the already existing implementations for the sake of easier feature addition. This eventually lead to version 2.0 of the `ViennaRNA PAckage`. Since then, he (re-)implemented a large portion of the currently existing library features, such as the new, generalized constraints framework, RNA folding grammar domain extensions, and the major part of the scripting language interface. Below is a list of most people who contributed more or less large portions of the implementations:

- Gregor Entzian (neighbor, walk)

- Mario Koestl (worked on SWIG interface and related unit testing)

- Dominik Luntzer (pertubation fold)

- Stefan Badelt (cofold evaluation, RNAdesign.pl, cofold findpath extensions)

- Stefan Hammer (parts of SWIG interface and corresponding unit tests)

- Ronny Lorenz (circfold, version 2.0, generic constraints, grammar extensions, and much more)

- Hakim Tafer (RNAplex, RNAsnoop)

- Ulrike Mueckstein (RNAup)

- Stephan Bernhart (cofold, plfold, unpaired probabilities, alifold, and so many more)

- Ivo Hofacker, Peter Stadler, and Christoph Flamm (almost every implementation up to version 1.8.5)

We also want to thank the following people:

- Sebastian Bonhoeffer's implementation of partition function folding served as a precursor to our part_func.c

- Manfred Tacker hacked constrained folding into fold.c for the first time

- Martin Fekete made the first attempts at "alignment folding"

- Andrea Tanzer and Martin Raden (Mann) for not stopping to report bugs found through comprehensive usage of our applications and RNAlib

- Thanks also to everyone else who helped testing and finding bugs, especially Christoph Flamm, Martijn Huynen, Baerbel Krakhofer, and many more

# Chapter 2

# Getting Started

- Installation and Configuration describes how to install and configure `RNAlib` for your requirements

- HelloWorld presents some small example programs to get a first impression on how to use this library

- HelloWorld (Perl/Python) contains small examples that show how to use RNAlib even without C/C++ programming skills from within your favorite scripting language

## 2.1 Installation and Configuration

A documentation on how to configure the different features of `RNAlib`, how to install the ViennaRNA Package, and finally, how to link you own programs against `RNAlib`.

### 2.1.1 Installing the ViennaRNA Package

For best portability the ViennaRNA package uses the GNU autoconf and automake tools. The instructions below are for installing the ViennaRNA package from source. However, pre-compiled binaries for various Linux distributions, as well as for Windows users are available from Download section of the `main ViennaRNA homepage`.

#### 2.1.1.1 Quick-start

Usually you'll just unpack, configure and make. To do this type:

```
tar -zxvf ViennaRNA-2.4.6.tar.gz
cd ViennaRNA-2.4.6
./configure
make
sudo make install
```

#### 2.1.1.2 Installation without root privileges

If you do not have root privileges on your computer, you might want to install the ViennaRNA Package to a location where you actually have write access to. To do so, you can set the installation prefix of the ./configure script like so:

```
./configure --prefix=/home/username/ViennaRNA
make install
```

This will install the entire ViennaRNA Package into a new directory ViennaRNA directly into the users username home directory.

**2.1.1.3    Notes for MacOS X users**

Although users will find /usr/bin/gcc and /usr/bin/g++ executables in their directory tree, these programs are not at all what they pretend to be. Instead of including the GNU programs, Apple decided to install clang/llvm in disguise. Unfortunately, the default version of clang/llvm does not support OpenMP (yet), but only complains at a late stage of the build process when this support is required. Therefore, it seems necessary to deactivate OpenMP support by passing the option –disable-openmp to the ./configure script.

Additionally, since MacOS X 10.5 the perl and python installation distributed with MacOS X always include so called universal-binaries (a.k.a. fat-binaries), i.e. binaries for multiple architecture types. In order to compile and link the programs, library, and scripting language interfaces of the ViennaRNA Package for multiple architectures, we've added a new configure switch that sets up the required changes automatically:

```
./configure --enable-universal-binary
```

**Note**

> Note, that with link time optimization turned on, MacOS X's default compiler (llvm/clang) generates an intermediary binary format that can not easily be combined into a multi-architecture library. Therefore, the –enable-universal-binary switch turns off link time optimization!

## 2.1.2    Configuring RNAlib features

The ViennaRNA Package includes additional executable programs such as RNAforester, Kinfold, and Kinwalker. Furthermore, we include several features in our C-library that may be activated by default, or have to be explicitly turned on at configure-time. Below we list a selection of the available configure options that affect the features included in all executable programs, the RNAlib C-library, and the corresponding scripting language interface(s).

**2.1.2.1    Streaming SIMD Extension (SSE) support**

Since version 2.3.5 our sources contain code that implements a faster multibranch loop decomposition in global MFE predictions, as used e.g. in RNAfold. This implementation makes use of modern processors capability to execute particular instructions on multiple data simultaneously (SIMD - single instruction multiple data, thanks to W. B. Langdon for providing the modified code). Consequently, the time required to assess the minimum of all multibranch loop decompositions is reduced up to about one half compared to the runtime of the original implementation. To make use of this piece of code you need a CPU capable to handle SSE4.1 instructions and enable the feature at compile-time using the following configure flag:

```
./configure --enable-sse
```

**2.1.2.2    Scripting Interfaces**

The ViennaRNA Package comes with scripting language interfaces for Perl 5, Python 2, and Python 3 (provided by swig), that allow one to use the implemented algorithms directly without the need of calling an executable program. The interfaces are build by default whenever the autoconf tool-chain detects the required build tools on your system. You may, however, explicitly turn off particular scripting language interface support at configure-time, for instance for Perl 5 and Python 2, before the actual installation.

Example:

```
./configure --without-perl --without-python
```

Disabling the scripting language support all-together can be accomplished using the following switch:

```
./configure --without-swig
```

### 2.1.2.3   Cluster Analysis

The programs AnalyseSeqs and AnalyseDists offer some cluster analysis tools (split decomposition, statistical geometry, neighbor joining, Ward's method) for sequences and distance data. To also build these programs add

```
--with-cluster
```

to your configure options.

### 2.1.2.4   Kinfold

The Kinfold program can be used to simulate the folding dynamics of an RNA molecule, and is compiled by default. Use the

```
--without-kinfold
```

option to skip compilation and installation of Kinfold.

### 2.1.2.5   RNAforester

The RNAforester program is used for comparing secondary structures using tree alignment. Similar to Kinfold, use the

```
--without-forester
```

option to skip compilation and installation of RNAforester.

### 2.1.2.6   Kinwalker

The Kinwalker algorithm performs co-transcriptional folding of RNAs, starting at a user specified structure (default↩
: open chain) and ending at the minimum free energy structure. Compilation and installation of this program is deactivated by default. Use the

```
--with-kinwalker
```

option to enable building and installation of Kinwalker.

### 2.1.2.7   Link Time Optimization (LTO)

To increase the performance of our implementations, the ViennaRNA Package tries to make use of the Link Time Optimization (LTO) feature of modern C-compilers. If you are experiencing any troubles at make-time or run-time, or the configure script for some reason detects that your compiler supports this feature although it doesn't, you can deactivate it using the flag

```
./configure --disable-lto
```

Note, that GCC before version 5 is known to produce unreliable LTO code, especially in combination with SSE (see Streaming SIMD Extension (SSE) support). We therefore recommend using a more recent compiler (GCC 5 or above) or to turn off one of the two features, LTO or SSE optimized code.

### 2.1.2.8 OpenMP support

To enable concurrent computation of our implementations and in some cases parallelization of the algorithms we make use of the OpenMP API. This interface is well understood by most modern compilers. However, in some cases it might be necessary to deactivate OpenMP support and therefore transform *RNAlib* into a C-library that is not entirely *thread-safe*. To do so, add the following configure option

```
./configure --disable-openmp
```

### 2.1.2.9 POSIX threads (pthread) support

To enable concurrent computation of multiple input data in RNAfold, and for our implementation of the concurrent unordered insert, ordered output flush data structure vrna_ostream_t we make use of POSIX threads. This should be supported on all modern platforms and usually does not pose any problems. Unfortunately, we use a threadpool implementation that is not compatible with Microsoft Windows yet. Thus, POSIX thread support can not be activated for Windows builds until we have fixed this problem. If you want to compile RNAfold and RNAlib without POSIX threads support for any other reasons, add the following configure option

```
./configure --disable-pthreads
```

### 2.1.2.10 Stochastic backtracking using Boustrophedon scheme

Stochastic backtracking for single RNA sequences, e.g. available through the RNAsubopt program, received a major speedup by implementing a Boustrophedon scheme (see this article for details). If for some reason you want to deactivate this feature, you can do that by adding the following switch to the configure script:

```
./configure --disable-boustrophedon
```

### 2.1.2.11 SVM Z-score filter in RNALfold

By default, RNALfold that comes with the ViennaRNA Package allows for z-score filtering of its predicted results using a support vector machine (SVM). However, the library we use to implement this feature (libsvm) is statically linked to our own RNAlib. If this introduces any problems for your own third-party programs that link against RNAlib, you can safely switch off the z-scoring implementation using

```
./configure --without-svm
```

### 2.1.2.12 GNU Scientific Library

The new program RNApvmin computes a pseudo-energy perturbation vector that aims to minimize the discrepancy of predicted, and observed pairing probabilities. For that purpose it implements several methods to solve the optimization problem. Many of them are provided by the GNU Scientific Library, which is why the RNApvmin program, and the RNAlib C-library are required to be linked against libgsl. If this introduces any problems in your own third-party programs that link against RNAlib, you can turn off a larger portion of available minimizers in RN←Apvmin and linking against libgsl all-together, using the switch

```
./configure --without-gsl
```

**2.1.2.13   Disable C11/C++11 feature support**

By default, we use C11/C++11 features in our implementations. This mainly accounts for unnamed unions/structs within *RNAlib*. The configure script automatically detects whether or not your compiler understands these features. In case you are using an older compiler, these features will be deactivated by setting a specific pre-processor directive. If for some reason you want to deactivate C11/C++11 features despite the capabilities of your compiler, use the following configure option:

```
./configure --disable-c11
```

**2.1.2.14   Enable warnings for use of deprecated symbols**

Since version 2.2 we are in the process of transforming the API of our *RNAlib*. Hence, several symbols are marked as *deprecated* whenever they have been replaced by the new API. By default, deprecation warnings at compile time are deactivated. If you want to get your terminal spammed by tons of deprecation warnings, enable them using:

```
./configure --enable-warn-deprecated
```

**2.1.2.15   Single precision partition function**

Calculation of partition functions (via RNAfold -p) uses double precision floats by default, to avoid overflow errors on longer sequences. If your machine has little memory and you don't plan to fold sequences over 1000 bases in length you can compile the package to do the computations in single precision by running

```
./configure --enable-floatpf
```

**Note**

> Using this option is discouraged and not necessary on most modern computers.

**2.1.2.16   Help**

For a complete list of all ./configure options and important environment variables, type

```
./configure --help
```

For more general information on the build process see the INSTALL.configure file.

**2.1.3   Linking against RNAlib**

In order to use our implemented algorithms you simply need to link your program to our *RNAlib* C-library that usually comes along with the ViennaRNA Package installation. If you've installed the ViennaRNA Package as a pre-build binary package, you probably need the corresponding development package, e.g. *viennarna-devel*, or *viennarna-dev*. The only thing that is left is to include the ViennaRNA header files into your source code, e.g.:

```
#include <ViennaRNA/mfe.h>
```

and start using our fast and efficient algorithm implementations.

**See also**

> In the mp_example and Some Examples using RNAlib API v3.0 sections, we list a small set of example code that usually is a good starting point for your application.

### 2.1.3.1 Compiler and Linker flags

Of course, simply adding the ViennaRNA header files into your source code is usually not enough. You probably need to tell your compiler where to find the header files, and sometimes add additional pre-processor directives. Whenever your installation of *RNAlib* was build with default settings and the header files were installed into their default location, a simple

```
-I/usr/include
```

pre-processor/compile flag should suffice. It can even be omitted in this case, since your compiler should search this directory by default anyway. You only need to change the path from */usr/include* to the correct location whenever the header files have been installed into a non-standard directory.

On the other hand, if you've compiled *RNAlib* with some non-default settings then you probably need to define some additional pre-processor macros:

- *VRNA_DISABLE_C11_FEATURES* . . . Disable C11/C++11 features.

  **Warning**

  > Add this directive to your pre-processor/compile flags only if *RNAlib* was build with the *–disable-c11* configure option.

  **See also**

  > Disable C11/C++11 feature support and vrna_C11_features()

- *VRNA_WARN_DEPRECATED* . . . Enable warnings for using deprecated symbols.

  **Note**

  > Adding this directive enables compiler warnings whenever you use symbols in *RNAlib* that are marked *deprecated*.

  **See also**

  > Enable warnings for use of deprecated symbols and Deprecated List

- *USE_FLOAT_PF* . . . Use single precision floating point operations instead of double precision in partition function computations.

  **Warning**

  > Define this macro only if *RNAlib* was build with the *–enable-floatpf* configure option!

  **See also**

  > Single precision partition function

Simply add the corresponding definition(s) to your pre-processor/compile flags, for instance:

```
-DVRNA_DISABLE_C11_FEATURES
```

Finally, linking against *RNAlib* is achieved by adding the following linker flag

```
-L/usr/lib -lRNA -fopenmp
```

Again, the path to the library, */usr/lib*, may be omitted if this path is searched for libraries by default. The second flag tells the linker to include *libRNA.a*, and the remaining two flags activate Link Time Optimization (LTO) and OpenMP support support, respectively.

**Note**

> Depending on your linker, the last two flags may differ.
> Depending on your configure time decisions, you can drop one or both of the last flags.
> In case you've compiled *RNAlib* with LTO support (See Link Time Optimization (LTO)) and you are using the
> same compiler for your third-party project that links against our library, you may add the
>
> ```
> -flto
> ```
>
> flag to enable Link Time Optimization.

#### 2.1.3.2 The pkg-config tool

Instead of hard-coding the required compiler and linker flags, you can also let the *pkg-config* tool automatically
determine the required flags. This tool is usually packaged for any Linux distribution and should be available for
MacOS X and MinGW as well. We ship a file *RNAlib2.pc* which is installed along with the static *libRNA.a* C-library
and populated with all required compiler and linker flags that correspond to your configure time decisions.

The compiler flags required for properly building your code that uses *RNAlib* can be easily obtained via

```
pkg-config --cflags RNAlib2
```

You get the corresponding linker flags using

```
pkg-config --libs RNAlib2
```

With this widely accepted standard it is also very easy to integrate *RNAlib* in your *autotools* project, just have a look
at the *PKG_CHECK_MODULES* macro.

## 2.2 HelloWorld

Below, you'll find some more or less simple C programs showing first steps into using *RNAlib*. A complete list of
example C programs can be found in the C Examples section.

**Simple MFE prediction for a given sequence**

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* The RNA sequence */
  char   *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

  /* allocate memory for MFE structure (length + 1) */
  char   *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* predict Minmum Free Energy and corresponding secondary structure */
  float mfe = vrna_fold(seq, structure);

  /* print sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

  /* cleanup memory */
  free(structure);

  return 0;
}
```

**See also**

> `examples/helloworld_mfe.c` in the source code tarball

### Simple MFE prediction for a multiple sequence alignment

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/alifold.h>
#include <ViennaRNA/aln_util.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* The RNA sequence alignment */
  const char  *sequences[] = {
    "CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGU",
    NULL /* indicates end of alignment */
  };

  /* compute the consensus sequence */
  char        *cons = consensus(sequences);

  /* allocate memory for MFE consensus structure (length + 1) */
  char        *structure = (char *)vrna_alloc(sizeof(char) * (strlen(sequences[0]) + 1));

  /* predict Minmum Free Energy and corresponding secondary structure */
  float       mfe = vrna_alifold(sequences, structure);

  /* print consensus sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", cons, structure, mfe);

  /* cleanup memory */
  free(cons);
  free(structure);

  return 0;
}
```

**See also**

> examples/helloworld_mfe_comparative.c in the source code tarball

### Simple Base Pair Probability computation

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* The RNA sequence */
  char        *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

  /* allocate memory for pairing propensity string (length + 1) */
  char        *propensity = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* pointers for storing and navigating through base pair probabilities */
  vrna_ep_t *ptr, *pair_probabilities = NULL;

  float       en = vrna_pf_fold(seq, propensity, &pair_probabilities);

  /* print sequence, pairing propensity string and ensemble free energy */
  printf("%s\n%s [ %6.2f ]\n", seq, propensity, en);

  /* print all base pairs with probability above 50% */
  for (ptr = pair_probabilities; ptr->i != 0; ptr++)
    if (ptr->p > 0.5)
      printf("p(%d, %d) = %g\n", ptr->i, ptr->j, ptr->p);

  /* cleanup memory */
  free(pair_probabilities);
  free(propensity);

  return 0;
}
```

**See also**

> `examples/helloworld_probabilities.c` in the source code tarball

**Deviating from the Default Model**

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/mfe.h>
#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/model.h>
#include <ViennaRNA/string_utils.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* initialize random number generator */
  vrna_init_rand();

  /* Generate a random sequence of 50 nucleotides */
  char       *seq = vrna_random_string(50, "ACGU");

  /* allocate memory for MFE structure (length + 1) */
  char       *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* create a new model details structure to store the Model Settings */
  vrna_md_t md;

  /* ALWAYS set default model settings first! */
  vrna_md_set_default(&md);

  /* change temperature and activate G-Quadruplex prediction */
  md.temperature  = 25.0; /* 25 Deg Celcius */
  md.gquad        = 1;    /* Turn-on G-Quadruples support */

  /* create a fold compound */
  vrna_fold_compound_t  *fc = vrna_fold_compound(seq, &md,
      VRNA_OPTION_DEFAULT);

  /* predict Minmum Free Energy and corresponding secondary structure */
  float                 mfe = vrna_mfe(fc, structure);

  /* print sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

  /* cleanup memory */
  free(structure);
  vrna_fold_compound_free(fc);

  return 0;
}
```

**See also**

> `examples/fold_compound_md.c` in the source code tarball

## 2.3 HelloWorld (Perl/Python)

### 2.3.1 Perl5

**Simple MFE prediction for a given sequence**

```perl
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

# compute minimum free energy (MFE) and corresponding structure
my ($ss, $mfe) = RNA::fold($seq);

# print output
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

### Simple MFE prediction for a multiple sequence alignment

```
use RNA;

# The RNA sequence alignment
my @sequences = (
    "CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGU"
);

# compute the consensus sequence
my $cons = RNA::consensus(\@sequences);

# predict Minmum Free Energy and corresponding secondary structure
my ($ss, $mfe) = RNA::alifold(\@sequences);

# print output
printf "%s\n%s [ %6.2f ]\n", $cons, $ss, $mfe;
```

### Deviating from the Default Model

```
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

# create a new model details structure
my $md = new RNA::md();

# change temperature and dangle model
$md->{temperature} = 20.0; # 20 Deg Celcius
$md->{dangles}     = 1;    # Dangle Model 1

# create a fold compound
my $fc = new RNA::fold_compound($seq, $md);

# predict Minmum Free Energy and corresponding secondary structure
my ($ss, $mfe) = $fc->mfe();

# print sequence, structure and MFE
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

## 2.3.2 Python

### Simple MFE prediction for a given sequence

```
import RNA

# The RNA sequence
seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA"

# compute minimum free energy (MFE) and corresponding structure
(ss, mfe) = RNA.fold(seq)

# print output
print "%s\n%s [ %6.2f ]" % (seq, ss, mfe)
```

### Simple MFE prediction for a multiple sequence alignment

```
import RNA

# The RNA sequence alignment
sequences = [
    "CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGU"
]

# compute the consensus sequence
cons = RNA.consensus(sequences)

# predict Minmum Free Energy and corresponding secondary structure
(ss, mfe) = RNA.alifold(sequences);

# print output
print "%s\n%s [ %6.2f ]" % (cons, ss, mfe)
```

**Deviating from the Default Model**

```python
import RNA

# The RNA sequence
seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA"

# create a new model details structure
md = RNA.md()

# change temperature and dangle model
md.temperature = 20.0 # 20 Deg Celcius
md.dangles     = 1    # Dangle Model 1

# create a fold compound
fc = RNA.fold_compound(seq, md)

# predict Minmum Free Energy and corresponding secondary structure
(ss, mfe) = fc.mfe()

# print sequence, structure and MFE
print "%s\n%s [ %6.2f ]\n" % (seq, ss, mfe)
```

# Chapter 3

# Concepts and Algorithms

This is an overview of the concepts and algorithms for which implementations can be found in this library.

Almost all of them rely on the physics based Nearest Neighbor Model for RNA secondary structure prediction.

- RNA Structure gives an introduction into the different layers of abstraction for RNA structures

- Distance Measures introduces different metrics to allow for the comparison of secondary structures

- Free Energy of Secondary Structures shows how the stability of a secondary structure can be quantified in terms of free energy

- Secondary Structure Folding Grammar explains the basic recursive decomposition scheme that is applied in secondary structure prediction

- RNA Secondary Structure Landscapes describes how transition paths between secondary structures span a landscape like graph

- Minimum Free Energy Algorithm(s) compute the most stable conformation in thermodynamic equilibrium

- Partition Function and Equilibrium Probabilitiy Algorithm(s) enable one to apply statistical mechanics to derive equilibrium probabilities of structure features

- Suboptimals and (other) Representative Structures allow for alternative description and enumeration of the structure ensemble

- RNA-RNA Interaction introduces how to model the interaction between RNA molecules

- Locally Stable Secondary Structures offer insights into structuredness of long sequences and entire genomes

- Comparative Structure Prediction augment structure prediction with evolutionary conservation of homologous sequences

- Classified DP variations perform an *a priori* partitioning of the structure ensemble and compute various properties for the resulting classes.

- RNA Sequence Design constitutes the inverse problem of structure prediction

- Experimental Structure Probing Data can be used to guide structure prediction, for instance using SHAPE reactivity data

- Ligand Binding adds more complexity to structure prediction by modelling the interaction between small chemical compounds or proteins and the RNA

- (Tertiary) Structure Motifs extend the abstraction of secondary structure beyond canonical base pair formation

## 3.1  RNA Structure

### 3.1.1  RNA Structures

### 3.1.2  Levels of Structure Abstraction

#### 3.1.2.1  Primary Structure

#### 3.1.2.2  Secondary Structure

#### 3.1.2.3  Tertiary Structure

#### 3.1.2.4  Quarternary Structure

#### 3.1.2.5  Pseudo-Knots

## 3.2  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)
```

Compute the "base pair" distance between two secondary structures s1 and s2.

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 [18]. All distance functions use the following global variables:

```
int   cost_matrix;
```

Specify the cost matrix to be used for distance calculations.

```
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.

**See also**

utils.h, dist_vars.h and stringdist.h for more details

### 3.2.1   Functions for Tree Edit Distances

```
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   free_tree(Tree *t)
```

Free the memory allocated for Tree t.

**See also**

dist_vars.h and treedist.h for prototypes and more detailed descriptions

### 3.2.2 Functions for String Alignment

```
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.

**See also**

> dist_vars.h and stringdist.h for prototypes and more detailed descriptions

### 3.2.3 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)
```

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

```
float profile_edit_distance ( const float *T1,
                              const float *T2)
```

Align the 2 probability profiles T1, T2
.

**See also**

> ProfileDist.h for prototypes and more details of the above functions

## 3.3   Free Energy of Secondary Structures
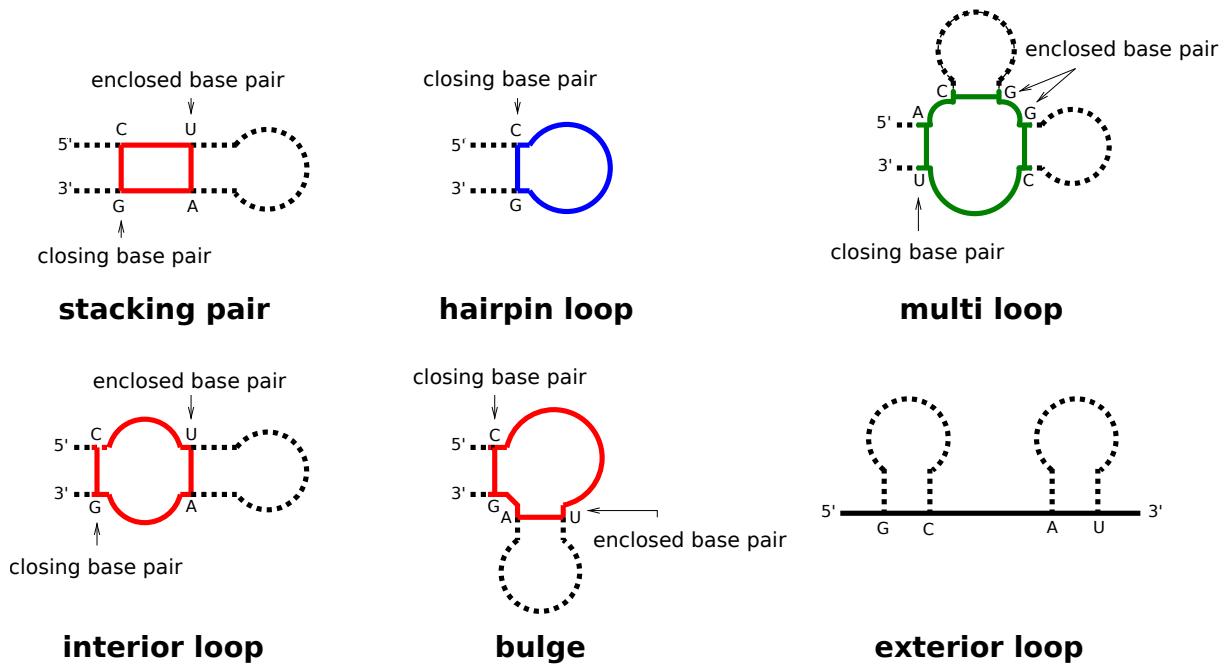
A description on how secondary structures are decomposed into individual loops to eventually evaluate their stability in terms of free energy.

### 3.3.1  Secondary Structure Loop Decomposition
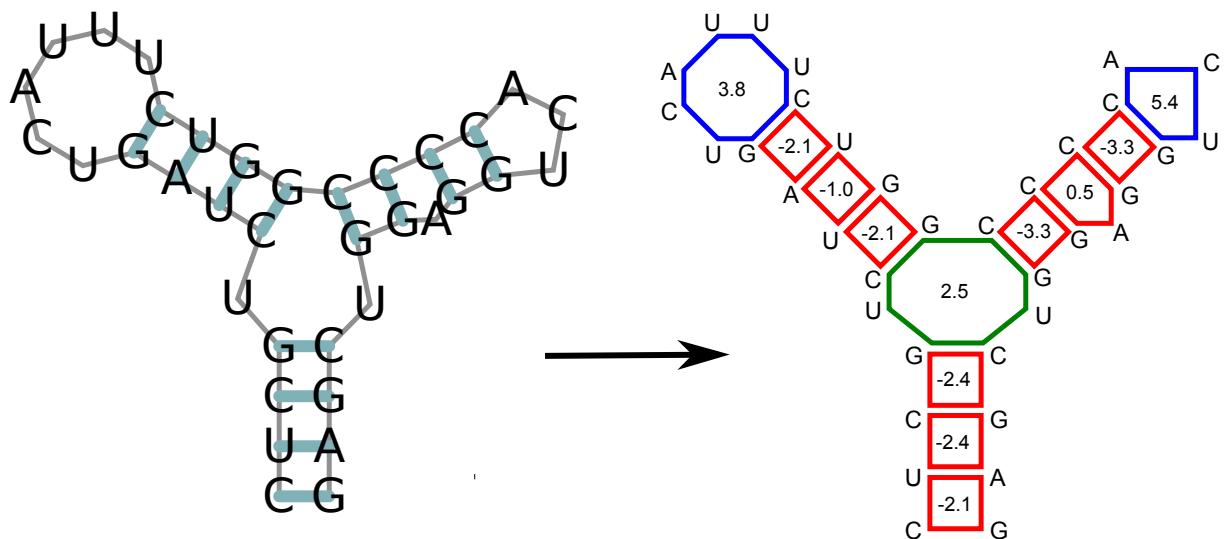
Each base pair in a secondary structure closes a loop, thereby directly enclosing unpaired nucleotides, and/or further base pairs. Our implementation distinguishes four basic types of loops:

- hairpin loops

- interior loops

- multibranch loops

- exterior loop

While the exterior loop is a special case without a closing pair, the other loops are determined by the number of base pairs involved in the loop formation, i.e. hairpin loops are 1-loops, since only a single base pair delimits the loop. interior loops are 2-loops due to their enclosing, and enclosed base pair. All loops where more than two base pairs are involved, are termed multibranch loops.



**stacking pair**                    **hairpin loop**                    **multi loop**



**interior loop**                    **bulge**                    **exterior loop**

Any secondary structure can be decomposed into its loops. Each of the loops then can be scored in terms of free energy, and the free energy of an entire secondary structure is simply the sum of free energies of its loops.

**3.3.1.1 Free Energy Evaluation API**

While we implement some functions that decompose a secondary structure into its individual loops, the majority of methods provided in are dedicated to free energy evaluation. The corresponding modules are:

**See also**

> Free energy evaluation, Process and evaluate individual loops

### 3.3.2 Free Energy Parameters

For secondary structure free energy evaluation we usually utilize the set of Nearest Neighbor Parameters also used in other software, such as *UNAfold* and *RNAstructure*. While the *RNAlib* already contains a compiled-in set of the latest *Turner 2004 Free Energy Parameters*, we defined a file format that allows to change these parameters at runtime. The `ViennaRNA Package` already comes with a set of parameter files containing

- Turner 1999 RNA parameters

- Mathews 1999 DNA parameters

- Andronescu 2007 RNA parameters

- Mathews 2004 DNA parameters

**3.3.2.1 Free Energy Parameters Modification API**

**See also**

> Energy parameters, Reading/Writing Energy Parameter Sets from/to File

### 3.3.3 Fine-tuning of the Energy Evaluation Model

**See also**
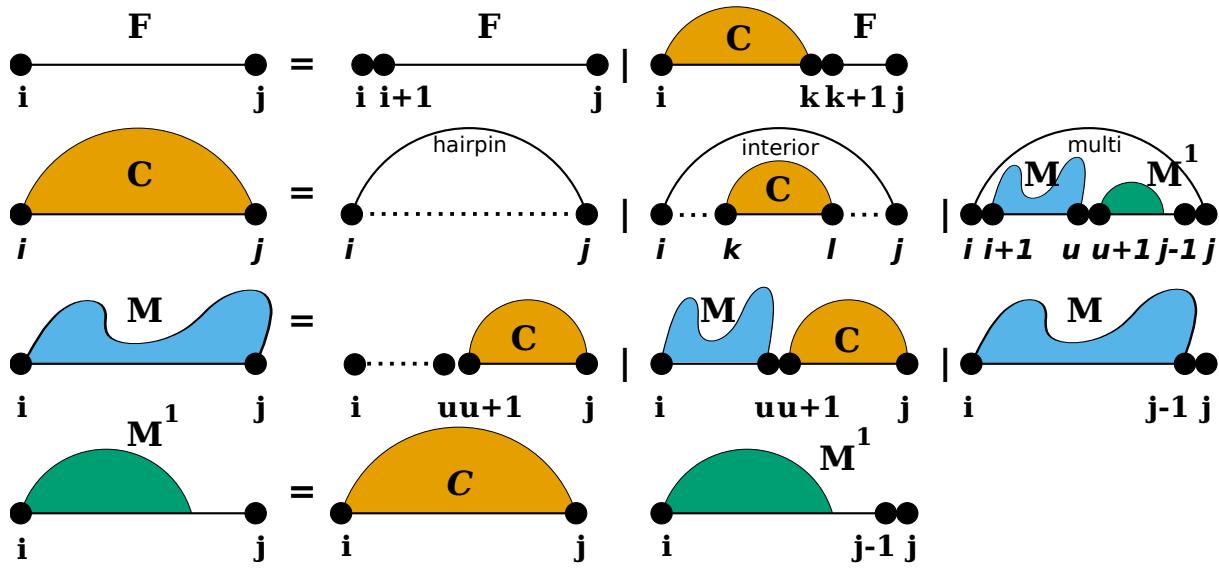
> Fine-tuning of the implemented models

## 3.4 Secondary Structure Folding Grammar

A description of the basic grammar to generate secondary structures, used for almost all prediction algorithms in our library and how to modify it.

### 3.4.1 Secondary Structure Folding Recurrences
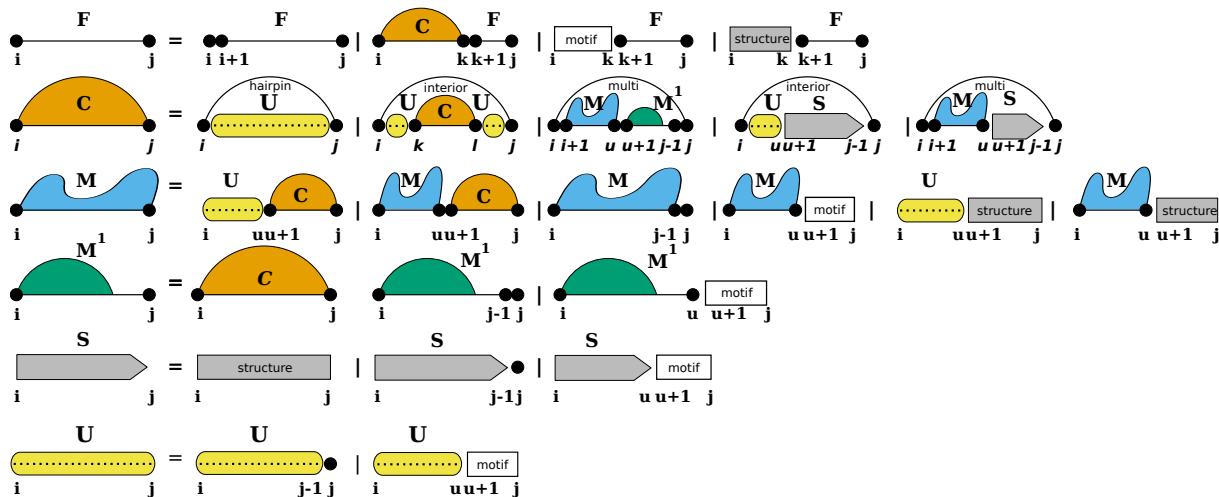
To predict secondary structures composed of the four distinguished loop types introduced before, all algorithms implemented in *RNAlib* follow a specific decomposition scheme, also known as the *RNA folding grammar*, or *Secondary Structure Folding Recurrences*.



However, compared to other RNA secondary structure prediction libraries, our implementation allows for a fine-grained control of the above recursions by constraining both, the individual derivations of the grammar as well as the evaluation of particular loop contributions. Furthermore, we provide a mechanism to extend the above grammar with additional derivation rules, so-called *Domains*.

### 3.4.2 Additional Structural Domains

Some applications of RNA secondary structure prediction require an extension of the *regular RNA folding grammar*. For instance one would like to include proteins and other ligands binding to unpaired loop regions while competing with conventional base pairing. Another application could be that one may want to include the formation of self-enclosed structural modules, such as *G-quadruplexes*. For such applications, we provide a pair of additional domains that extend the regular RNA folding grammar, Structured domains and Unstructured domains.



While unstructured domains are usually determined by a more or less precise sequence motif, e.g. the binding site for a protein, structured domains are considered self-enclosed modules with a more or less complex pairing pattern. Our extension with these two domains introduces two production rules to fill additional dynamic processing matrices S and U where we store the pre-computed contributions of structured domains (S), and unstructured domains (U).

### 3.4.2.1 Structured Domains
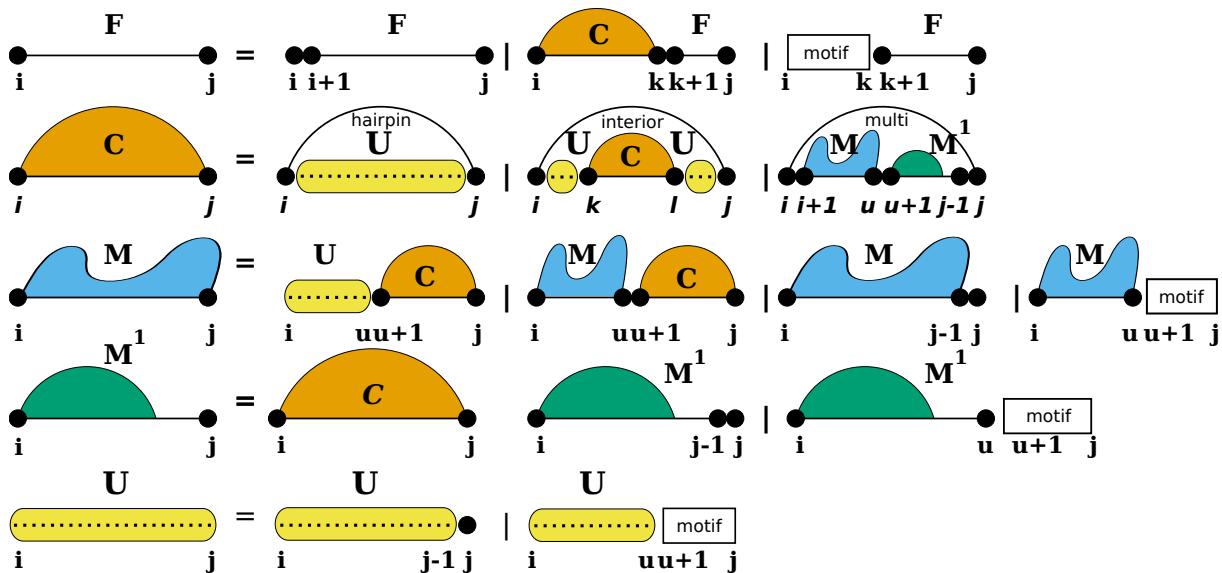
Usually, structured domains represent self-enclosed structural modules that exhibit a more or less complex base pairing pattern. This can be more or less well-defined 3D motifs, such as *G-Quadruplexes*, or loops with additional non-canonical base pair interactions, such as *kink-turns*.

**Note**

> Currently, our implementation only provides the specialized case of *G-Quadruplexes*.

### 3.4.2.2 Unstructured Domains

Unstructured domains appear in the production rules of the RNA folding grammar wherever new unpaired nucleotides are attached to a growing substructure (see also [12]):



The white boxes represent the stretch of RNA bound to the ligand and represented by a more or less specific sequence motif. The motif itself is considered unable to form base pairs. The additional production rule *U* is used to precompute the contribution of unpaired stretches possibly bound by one or more ligands. The auxiliary DP matrix for this production rule is filled right before processing the other (regular) production rules of the RNA folding grammar.

### 3.4.2.3 Domain Extension API

For the sake of flexibility, each of the domains is associated with a specific data structure serving as an abstract interface to the extension. The interface uses callback functions to

- pre-compute arbitrary data, e.g. filling up additional dynamic programming matrices, and

- evaluate the contribution of a paired or unpaired structural feature of the RNA.

Implementations of these callbacks are separate for regular free energy evaluation, e.g. MFE prediction, and partition function applications. A data structure holding arbitrary data required for the callback functions can be associated to the domain as well. While *RNAlib* comes with a default implementation for structured and unstructured domains, the system is entirely user-customizable.

**See also**

> Unstructured domains, Structured domains, G-quadruplexes, Ligands binding to unstructured domains

### 3.4.3    Constraints on the Folding Grammar

Secondary Structure constraints can be subdivided into two groups:

- Hard Constraints

- Soft Constraints

While Hard-Constraints directly influence the production rules used in the folding recursions by allowing, disallowing, or enforcing certain decomposition steps, Soft-constraints on the other hand are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations.

**Note**

> Secondary structure constraints are always applied at decomposition level, i.e. in each step of the recursive structure decomposition, for instance during MFE prediction.

#### 3.4.3.1    Hard Constraints API

Hard constraints as implemented in our library can be specified for individual loop types, i.e. the atomic derivations of the RNA folding grammar rules. Hence, the pairing behavior of both, single nucleotides and pairs of bases, can be constrained in every loop context separately. Additionally, an abstract implementation using a callback mechanism allows for full control of more complex hard constraints.

**See also**

> Hard constraints

#### 3.4.3.2    Soft Constraints API

For the sake of memory efficiency, we do not implement a loop context aware version of soft constraints. The *static* soft constraints as implemented only distinguish unpaired from paired nucleotides. This is usually sufficient for most use-case scenarios. However, similar to hard constraints, an abstract soft constraints implementation using a callback mechanism exists, that allows for any soft constraint that is compatible with the RNA folding grammar. Thus, loop contexts and even individual derivation rules can be addressed separately for maximum flexibility in soft-constraints application.

**See also**

> Soft constraints,      Incorporating ligands binding to specific sequence/structure motifs using soft constraints, SHAPE reactivity data

## 3.5    RNA Secondary Structure Landscapes

A description of the implicit landscape-like network of structures that appears upon modelling the transition of one structure into another

### 3.5.1  The Neighborhood of a Secondary Structure

### 3.5.2  The Secondary Structure Landscape API

## 3.6  Minimum Free Energy Algorithm(s)

Computing the Minimum Free Energy (MFE), i.e. the most stable conformation in thermodynamic equilibrium

### 3.6.1  Zuker's Algorithm

Our library provides fast dynamic programming Minimum Free Energy (MFE) folding algorithms derived from the decomposition scheme as described by "Zuker & Stiegler (1981)" [24].

### 3.6.2  MFE for circular RNAs

Folding of *circular* RNA sequences is handled as a post-processing step of the forward recursions.  See [9] for further details.

### 3.6.3  MFE Algorithm API

We provide interfaces for the prediction of

- MFE and corresponding secondary structure for single sequences,

- consensus MFE structures of sequence alignments, and

- MFE structure for two hybridized RNA strands

**See also**

Minimum Free Energy (MFE) algorithms, MFE Consensus Structures for Sequence Alignment(s), MFE Structures of two hybrid Calculating MFE representatives of a Distance Based Partitioning

## 3.7  Partition Function and Equilibrium Probabilitiy Algorithm(s)

### 3.7.1  Equilibrium Ensemble Statistics

In contrast to methods that compute the property of a single structure in the ensemble, e.g. Minimum Free Energy Algorithm(s), the partition function algorithms always consider the entire equilibrium ensemble. For that purpose, the McCaskill algorithm [15] and its variants can be used to efficiently compute

- the partition function, and from that

- various equilibrium probabilities, for instance base pair probabilities, probabilities of individual structure motifs, and many more.

The principal idea behind this approach is that in equilibrium, statistical mechanics and polymer theory tells us that the frequency or probability $p(s)$ of a particular state $s$ depends on its energy $E(s)$ and follows a Boltzmann distribution, i.e.

$$p(s) \propto e^{-\beta E(s)} \text{ with } \beta = \frac{1}{kT}$$

where $k \approx 1.987 \cdot 10^{-3} \frac{kcal}{mol\ K}$ is the Boltzmann constant, and $T$ the thermodynamic temperature. From that relation, the actual probability of state $s$ can then be obtained using a proper scaling factor, the *canonical partition function*

$$Z = \sum_{s \in \Omega} e^{-\beta E(s)}$$

where $\Omega$ is the finite set of all states. Finally, the equilibrium probability of state $s$ can be computed as

$$p(s) = \frac{e^{-\beta E(s)}}{Z}$$

Instead of enumerating all states exhaustively to compute $Z$ one can apply the Secondary Structure Folding Recurrences again for an efficient computation in cubic time. An *outside* variant of the same recursions is then used to compute probabilities for base pairs, stretches of consecutive unpaired nucleotides, or structural motifs.

**See also**

> Further details of the Partition function and Base Pair Probability algorithm can be obtained from McCaskill 1990 [15]

### 3.7.2 Partition Function and Equilibrium Probability API

We implement a wide variety of variants of the partition function algorithm according to McCaskill 1990 [15]. See the corresponding submodules for specific implementation details.

**See also**

> Partition function and equilibrium properties, Partition Function and Base Pair Probabilities for Sequence Alignment(s), Partition Function for two hybridized Sequences, Partition Function for two hybridized Sequences as a stepwise Process, Partition Function and Equilibrium Probabilities for Locally Stable Secondary Structures, Calculate Partition Functions of a Dist

## 3.8 Suboptimals and (other) Representative Structures

### 3.8.1 Suboptimal Secondary Structures

### 3.8.2 Sampling Secondary Structures from the Ensemble

### 3.8.3 Structure Enumeration and Sampling API

**See also**

> Suboptimal structures sensu Stiegler et al. 1984 / Zuker et al. 1989, Suboptimal structures within an energy band around the M
> Structure sampling from the ensemble, Compute the structure with maximum expected accuracy (MEA),
> Compute the centroid structure

## 3.9    RNA-RNA Interaction

### 3.9.1    rip_intro

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

### 3.9.2    rip_intro

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 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).

### 3.9.3    RNA-RNA interaction as a Stepwise Process

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.

### 3.9.4    RNA-RNA Interaction API

## 3.10    Locally Stable Secondary Structures

### 3.10.1    local_intro

### 3.10.2    local_mfe

### 3.10.3    local_pf

### 3.10.4    Locally Stable Secondary Structure API

## 3.11    Comparative Structure Prediction

### 3.11.1    Incorporate Evolutionary Information

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 [7] and [1].

### 3.11.2 Comparative Structure Prediction API

## 3.12 Classified DP variations

### 3.12.1 The Idea of Classified Dynamic Programming

Usually, thermodynamic properties using the basic recursions for Minimum Free Energy Algorithm(s), Partition Function and Equilibriu and so forth, are computed over the entire structure space. However, sometimes it is desired to partition the structure space *a priori* and compute the above properties for each of the resulting partitions. This approach directly leads to *Classified Dynamic Programming*.

### 3.12.2 Distance Class Partitioning

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, we refer to Lorenz et al. 2009 [11]

### 3.12.3 Density of States (DOS)

### 3.12.4 Classified DP API

## 3.13 RNA Sequence Design

### 3.13.1 Generate Sequences that fold into particular Secondary Structures

### 3.13.2 RNA Sequence Design API

**See also**

> Inverse Folding (Design)

## 3.14 Experimental Structure Probing Data

### 3.14.1 Guide the Structure Prediction using Experimental Data

#### 3.14.1.1 SHAPE reactivities

### 3.14.2 Structure Probing Data API

**See also**

> Experimental structure probing data, SHAPE reactivity data, pertubation

## 3.15 Ligand Binding

### 3.15.1 Small Molecules and Proteins that bind to specific RNA Structures

### 3.15.2 ligand_binding_api

In our library, we provide two different ways to incorporate ligand binding to RNA structures:

- Ligands binding to unstructured domains, and

- Incorporating ligands binding to specific sequence/structure motifs using soft constraints

The first approach is implemented as an actual extension of the folding grammar. It adds auxiliary derivation rules for each case when consecutive unpaired nucleotides are evaluated. Therefore, this model is applicable to ligand binding to any loop context.

The second approach, on the other hand, uses the soft-constraints feature to change the energy evaluation of hairpin- or interior-loops. Hence, it can only be appleid when a ligand binds to a hairpin-like, or interior-loop like motif.

**See also**

> Ligands binding to unstructured domains, Incorporating ligands binding to specific sequence/structure motifs using soft constra

## 3.16 (Tertiary) Structure Motifs

### 3.16.1 Incorporating Higher-Order (Tertiary) Structure Motifs

### 3.16.2 RNA G-Quadruplexes

### 3.16.3 (Tertiary) Structure Motif API

# Chapter 4

# I/O Formats

Below, you'll find a listing of different sections that introduce the most common notations of sequence and structure data, specifications of bioinformatics sequence and structure file formats, and various output file formats produced by our library.

- RNA Structure Notations describes the different notations and representations of RNA secondary structures

- File Formats gives an overview of the file formats compatible with our library

- Plotting shows the different (PostScript) plotting functions for RNA secondary structures, feature probabilities, and multiple sequence alignments

## 4.1  RNA Structure Notations

### 4.1.1  Representations of Secondary Structures

The standard representation of a secondary structure in our library is the Dot-Bracket Notation (a.k.a. Dot-Parenthesis Notation), where matching brackets symbolize base pairs and unpaired bases are shown as dots. Based on that notation, more elaborate representations have been developed to include additional information, such as the loop context a nucleotide belongs to and to annotated pseudo-knots.

**See also**

> Extended Dot-Bracket Notation, Washington University Secondary Structure (WUSS) notation

Alternatively, one may find representations with 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 [6] 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 [17], 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 '_'.

### 4.1.2   Dot-Bracket Notation (a.k.a. Dot-Parenthesis Notation)

The Dot-Bracket notation as introduced already in the early times of the ViennaRNA Package denotes base pairs by matching pairs of parenthesis `()` and unpaired nucleotides by dots `.`.

Example: A simple helix of size 4 enclosing a hairpin of size 4 is annotated as

```
((((....))))
```

**See also**

> [vrna_ptable_from_string()](#), [vrna_db_flatten()](#), [vrna_db_flatten_to()](#)

### 4.1.3   Extended Dot-Bracket Notation

A more generalized version of the original Dot-Bracket notation may use additional pairs of brackets, such as $<>$, $\{\}$, and $[\,]$, and matching pairs of uppercase/lowercase letters. This allows for anotating pseudo-knots, since different pairs of brackets are not required to be nested.

Example: The follwing annotations of a simple structure with two crossing helices of size 4 are equivalent:

```
<<<<[[[[....>>>>]]]]
((((AAAA....))))aaaa
AAAA{{{{....aaaa}}}}
```

**See also**

> [vrna_ptable_from_string()](#), [vrna_db_flatten()](#), [vrna_db_flatten_to()](#)

### 4.1.4  Washington University Secondary Structure (WUSS) notation

The WUSS notation, as frequently used for consensus secondary structures in Stockholm 1.0 format allows for a fine-grained annotation of base pairs and unpaired nucleotides, including pseudo-knots.

Below, you'll find a list of secondary structure elements and their corresponding WUSS annotation (See also the infernal user guide at `http://eddylab.org/infernal/Userguide.pdf`)

- **Base pairs**

  Nested base pairs are annotated by matching pairs of the symbols `<>`, `()`, `{}`, and `[]`. Each of the matching pairs of parenthesis have their special meaning, however, when used as input in our programs, e.g. structure constraint, these details are usually ignored. Furthermore, base pairs that constitute as pseudo-knot are denoted by letters from the latin alphabet and are, if not denoted otherwise, ignored entirely in our programs.

- **Hairpin loops**

  Unpaired nucleotides that constitute the hairpin loop are indicated by underscores, `_`.

  Example: `<<<<<_____>>>>>`

- **Bulges and interior loops**

  Residues that constitute a bulge or interior loop are denoted by dashes, `-`.

  Example: `(((-<<_____>>-)))`

- **Multibranch loops**

  Unpaired nucleotides in multibranch loops are indicated by commas `,`.

  Example: `(((,,<<_____>>,<<____>>)))`

- **External residues**

  Single stranded nucleotides in the exterior loop, i.e. not enclosed by any other pair are denoted by colons, `:`.

  Example: `<<<____>>>:::`

- **Insertions**

  In cases where an alignment represents the consensus with a known structure, insertions relative to the known structure are denoted by periods, `.`. Regions where local structural alignment was invoked, leaving regions of both target and query sequence unaligned, are indicated by tildes, `~`.

  **Note**

  > These symbols only appear in alignments of a known (query) structure annotation to a target sequence of unknown structure.

- **Pseudo-knots**

  The WUSS notation allows for annotation of pseudo-knots using pairs of upper-case/lower-case letters.

  **Note**

  > Our programs and library functions usually ignore pseudo-knots entirely treating them as unpaired nucleotides, if not stated otherwise.

  Example: `<<<_AAA___>>>aaa`

**See also**

vrna_db_from_WUSS()

### 4.1.5   Examples for Structure Parsing and Conversion

### 4.1.6   Structure Parsing and Conversion API

Several functions are provided for parsing structures and converting to different representations.

```
char  *expand_Full(const char *structure)
```

Convert the full structure from bracket notation to the expanded notation including root.

```
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  *expand_Shapiro (const char *coarse);
```

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from b2C().

```
char *add_root (const char *structure)
```

Adds a root to an un-rooted tree in any except bracket notation.

```
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.

**See also**

>    RNAstruct.h for prototypes and more detailed description

## 4.2   File Formats

### 4.2.1   File formats for Multiple Sequence Alignments (MSA)

#### 4.2.1.1   ClustalW format

The *ClustalW* format is a relatively simple text file containing a single multiple sequence alignment of DNA, RNA, or protein sequences. It was first used as an output format for the *clustalw* programs, but nowadays it may also be generated by various other sequence alignment tools. The specification is straight forward:

- The first line starts with the words

  ```
  CLUSTAL W
  ```

  or

  ```
  CLUSTALW
  ```

- After the above header there is at least one empty line

- Finally, one or more blocks of sequence data are following, where each block is separated by at least one empty line

Each line in a blocks of sequence data consists of the sequence name followed by the sequence symbols, separated by at least one whitespace character. Usually, the length of a sequence in one block does not exceed 60 symbols. Optionally, an additional whitespace separated cumulative residue count may follow the sequence symbols. Optionally, a block may be followed by a line depicting the degree of conservation of the respective alignment columns.

**Note**

> Sequence names and the sequences must not contain whitespace characters! Allowed gap symbols are the hyphen ("-"), and dot (".").

**Warning**

> Please note that many programs that output this format tend to truncate the sequence names to a limited number of characters, for instance the first 15 characters. This can destroy the uniqueness of identifiers in your MSA.

Here is an example alignment in ClustalW format:

```
CLUSTAL W (1.83) multiple sequence alignment

AL031296.1/85969-86120        CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGUAACAAUACUUAC
AANU01225121.1/438-603        CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGUGACAAUACUUAC
AAWR02037329.1/29294-29150    ---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGUAGUAGUACCAAU

AL031296.1/85969-86120        UCUCGUUGGUGAUAAGGAACAGCU
AANU01225121.1/438-603        UCUCGUUGGUGAUAAGGAACAGCU
AAWR02037329.1/29294-29150    GCUAAUUAGUUGUGAGGACCAACU
```

#### 4.2.1.2 Stockholm 1.0 format

Here is an example alignment in Stockholm 1.0 format:

```
# STOCKHOLM 1.0

#=GF AC   RF01293
#=GF ID   ACA59
#=GF DE   Small nucleolar RNA ACA59
#=GF AU   Wilkinson A
#=GF SE   Predicted; WAR; Wilkinson A
#=GF SS   Predicted; WAR; Wilkinson A
#=GF GA   43.00
#=GF TC   44.90
#=GF NC   40.30
#=GF TP   Gene; snRNA; snoRNA; HACA-box;
#=GF BM   cmbuild -F CM SEED
#=GF CB   cmcalibrate --mpi CM
#=GF SM   cmsearch --cpu 4 --verbose --nohmmonly -E 1000 -Z 549862.597050 CM SEQDB
#=GF DR   snoRNABase; ACA59;
#=GF DR   SO; 0001263; ncRNA_gene;
#=GF DR   GO; 0006396; RNA processing;
#=GF DR   GO; 0005730; nucleolus;
#=GF RN   [1]
#=GF RM   15199136
#=GF RT   Human box H/ACA pseudouridylation guide RNA machinery.
#=GF RA   Kiss AM, Jady BE, Bertrand E, Kiss T
#=GF RL   Mol Cell Biol. 2004;24:5797-5807.
#=GF WK   Small_nucleolar_RNA
#=GF SQ   3


AL031296.1/85969-86120     CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGUAACAAUACUUACUCUCGUUGGUGAUAAGGAACAGC
AANU01225121.1/438-603     CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGUGACAAUACUUACUCUCGUUGGUGAUAAGGAACAGC
AAWR02037329.1/29294-29150 ---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGUAGUAGUACCAAUGCUAAUUAGUUGUGAGGACCAAC
#=GC SS_cons               -----((((,<<<<<<<<<_____>>>>>>>>,,,,<<<<<<<_____>>>>>>,,,,,))))::::::::::::
#=GC RF                    CUGCcccaCAaCacuuguGCCUCaGUUACcCauagguGuAGUGaGgGuggcAaUACccaCcCucgUUgGuggUaAGGAaCAgC
//
```

**See also**

Washington University Secondary Structure (WUSS) notation on legal characters for the consensus secondary structure line *SS_cons* and their interpretation

#### 4.2.1.3 FASTA (Pearson) format

**Note**

Sequence names must not contain whitespace characters. Otherwise, the parts after the first whitespace will be dropped. The only allowed gap character is the hyphen ("-").

Here is an example alignment in FASTA format:

```
>AL031296.1/85969-86120
CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGUAACAAUACUUAC
UCUCGUUGGUGAUAAGGAACAGCU
>AANU01225121.1/438-603
CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGUGACAAUACUUAC
UCUCGUUGGUGAUAAGGAACAGCU
>AAWR02037329.1/29294-29150
---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGUAGUAGUACCAAU
GCUAAUUAGUUGUGAGGACCAACU
```

#### 4.2.1.4 MAF format

The multiple alignment format (MAF) is usually used to store multiple alignments on DNA level between entire genomes. It consists of independent blocks of aligned sequences which are annotated by their genomic location. Consequently, an MAF formatted MSA file may contain multiple records. MAF files start with a line

```
##maf
```

which is optionally extended by whitespace delimited key=value pairs. Lines starting with the character ("#") are considered comments and usually ignored.

A MAF block starts with character ("a") at the beginning of a line, optionally followed by whitespace delimited key=value pairs. The next lines start with character ("s") and contain sequence information of the form

```
s src start size strand srcSize sequence
```

where

- *src* is the name of the sequence source

- *start* is the start of the aligned region within the source (0-based)

- *size* is the length of the aligned region without gap characters

- *strand* is either ("+") or ("-"), depicting the location of the aligned region relative to the source

- *srcSize* is the size of the entire sequence source, e.g. the full chromosome

- *sequence* is the aligned sequence including gaps depicted by the hyphen ("-")

Here is an example alignment in MAF format (bluntly taken from the UCSC Genome browser website):

```
##maf version=1 scoring=tba.v8
# tba.v8 (((human chimp) baboon) (mouse rat))
# multiz.v7
# maf_project.v5 _tba_right.maf3 mouse _tba_C
# single_cov2.v4 single_cov2 /dev/stdin

a score=23262.0
s hg16.chr7    27578828 38 + 158545518 AAA-GGGAATGTTAACCAAATGA---ATTGTCTCTTACGGTG
s panTro1.chr6 28741140 38 + 161576975 AAA-GGGAATGTTAACCAAATGA---ATTGTCTCTTACGGTG
s baboon         116834 38 +   4622798 AAA-GGGAATGTTAACCAAATGA---GTTGTCTCTTATGGTG
s mm4.chr6     53215344 38 + 151104725 -AATGGGAATGTTAAGCAAACGA---ATTGTCTCTCAGTGTG
s rn3.chr4     81344243 40 + 187371129 -AA-GGGGATGCTAAGCCAATGAGTTGTTGTCTCTCAATGTG

a score=5062.0
s hg16.chr7    27699739 6 + 158545518 TAAAGA
s panTro1.chr6 28862317 6 + 161576975 TAAAGA
s baboon         241163 6 +   4622798 TAAAGA
s mm4.chr6     53303881 6 + 151104725 TAAAGA
s rn3.chr4     81444246 6 + 187371129 taagga

a score=6636.0
s hg16.chr7    27707221 13 + 158545518 gcagctgaaaaca
s panTro1.chr6 28869787 13 + 161576975 gcagctgaaaaca
s baboon         249182 13 +   4622798 gcagctgaaaaca
s mm4.chr6     53310102 13 + 151104725 ACAGCTGAAAATA
```

## 4.2.2 File formats to manipulate the RNA folding grammar

### 4.2.2.1 Command Files

The RNAlib and many programs of the ViennaRNA Package can parse and apply data from so-called command files. These commands may refer to structure constraints or even extensions of the RNA folding grammar (such as Unstructured domains). Commands are given as a line of whitespace delimited data fields. The syntax we use extends the constraint definitions used in the `mfold`/`UNAfold` software, where each line begins with a command character followed by a set of positions.
However, we introduce several new commands, and allow for an optional loop type context specifier in form of a sequence of characters, and an orientation flag that enables one to force a nucleotide to pair upstream, or downstream.

#### 4.2.2.1.1 Constraint commands

The following set of commands is recognized:

- `F` . . . Force

- `P` . . . Prohibit

- `C` . . . Conflicts/Context dependency

- `A` . . . Allow (for non-canonical pairs)

- `E` . . . Soft constraints for unpaired position(s), or base pair(s)

#### 4.2.2.1.2 RNA folding grammar exensions

- `UD` . . . Add ligand binding using the Unstructured domains feature

#### 4.2.2.1.3 Specification of the loop type context

The optional loop type context specifier [LOOP] may be a combination of the following:

- `E` . . . Exterior loop

- `H` . . . Hairpin loop

- `I` . . . Interior loop

- `M` . . . Multibranch loop

- `A` . . . All loops

For structure constraints, we additionally allow one to address base pairs enclosed by a particular kind of loop, which results in the specifier [WHERE] which consists of [LOOP] plus the following character:

- `i` . . . enclosed pair of an Interior loop

- `m` . . . enclosed pair of a Multibranch loop

If no [LOOP] or [WHERE] flags are set, all contexts are considered (equivalent to `A` )

#### 4.2.2.1.4 Controlling the orientation of base pairing

For particular nucleotides that are forced to pair, the following [ORIENTATION] flags may be used:

- `U` . . . Upstream

- `D` . . . Downstream

If no [ORIENTATION] flag is set, both directions are considered.

#### 4.2.2.1.5 Sequence coordinates

Sequence positions of nucleotides/base pairs are $1-$ based and consist of three positions $i$, $j$, and $k$. Alternativly, four positions may be provided as a pair of two position ranges $[i:j]$, and $[k:l]$ using the '-' sign as delimiter within each range, i.e. $i - j$, and $k - l$.

#### 4.2.2.1.6 Valid constraint commands

Below are resulting general cases that are considered *valid* constraints:

1. **"Forcing a range of nucleotide positions to be paired"**:
   Syntax:

   ```
   F i 0 k [WHERE] [ORIENTATION]
   ```

   Description:
   Enforces the set of $k$ consecutive nucleotides starting at position $i$ to be paired. The optional loop type specifier [WHERE] allows to force them to appear as closing/enclosed pairs of certain types of loops.

2. **"Forcing a set of consecutive base pairs to form"**:
   Syntax:

   ```
   F i j k [WHERE]
   ```

   Description:
   Enforces the base pairs $(i, j), \ldots, (i + (k-1), j - (k-1))$ to form. The optional loop type specifier [WHERE] allows to specify in which loop context the base pair must appear.

3. **"Prohibiting a range of nucleotide positions to be paired"**:
   Syntax:

   ```
   P i 0 k [WHERE]
   ```

   Description:
   Prohibit a set of $k$ consecutive nucleotides to participate in base pairing, i.e. make these positions unpaired. The optional loop type specifier [WHERE] allows to force the nucleotides to appear within the loop of specific types.

4. **"Probibiting a set of consecutive base pairs to form"**:
   Syntax:

   ```
   P i j k [WHERE]
   ```

Description:
Probibit the base pairs $(i, j), \ldots, (i + (k-1), j - (k-1))$ to form. The optional loop type specifier [WHERE] allows to specify the type of loop they are disallowed to be the closing or an enclosed pair of.

5. **"Prohibiting two ranges of nucleotides to pair with each other"**:
   Syntax:

   ```
   P i-j k-l [WHERE]
   ```

   Description:
   Prohibit any nucleotide $p \in [i : j]$ to pair with any other nucleotide $q \in [k : l]$. The optional loop type specifier [WHERE] allows to specify the type of loop they are disallowed to be the closing or an enclosed pair of.

6. **"Enforce a loop context for a range of nucleotide positions"**:
   Syntax:

   ```
   C i 0 k [WHERE]
   ```

   Description:
   This command enforces nucleotides to be unpaired similar to *prohibiting* nucleotides to be paired, as described above. It too marks the corresponding nucleotides to be unpaired, however, the [WHERE] flag can be used to enforce specfic loop types the nucleotides must appear in.

7. **"Remove pairs that conflict with a set of consecutive base pairs"**:
   Syntax:

   ```
   C i j k
   ```

   Description:
   Remove all base pairs that conflict with a set of consecutive base pairs $(i, j), \ldots, (i + (k-1), j - (k-1))$. Two base pairs $(i, j)$ and $(p, q)$ conflict with each other if $i < p < j < q$, or $p < i < q < j$.

8. **"Allow a set of consecutive (non-canonical) base pairs to form"**:
   Syntax:

   ```
   A i j k [WHERE]
   ```

   Description:
   This command enables the formation of the consecutive base pairs $(i, j), \ldots, (i + (k-1), j - (k-1))$, no matter if they are *canonical*, or *non-canonical*. In contrast to the above `F` and `W` commands, which remove conflicting base pairs, the `A` command does not. Therefore, it may be used to allow *non-canonical* base pair interactions. Since the RNAlib does not contain free energy contributions $E_{ij}$ for non-canonical base pairs $(i, j)$, they are scored as the *maximum* of similar, known contributions. In terms of a *Nussinov* like scoring function the free energy of non-canonical base pairs is therefore estimated as

$$E_{ij} = \min \left[ \max_{(i,k) \in \{GC, CG, AU, UA, GU, UG\}} E_{ik}, \max_{(k,j) \in \{GC, CG, AU, UA, GU, UG\}} E_{kj} \right].$$

The optional loop type specifier [WHERE] allows to specify in which loop context the base pair may appear.

9. **"Apply pseudo free energy to a range of unpaired nucleotide positions"**:
   Syntax:

   ```
   E i 0 k e
   ```

   Description:
   Use this command to apply a pseudo free energy of $e$ to the set of $k$ consecutive nucleotides, starting at position $i$. The pseudo free energy is applied only if these nucleotides are considered unpaired in the recursions, or evaluations, and is expected to be given in $kcal/mol$.

10. **"Apply pseudo free energy to a set of consecutive base pairs"**:
    Syntax

    ```
    E i j k e
    ```

    Use this command to apply a pseudo free energy of $e$ to the set of base pairs $(i, j), \ldots, (i + (k - 1), j - (k - 1))$. Energies are expected to be given in $kcal/mol$.

#### 4.2.2.1.7 Valid domain extensions commands

1. **"Add ligand binding to unpaired motif (a.k.a. unstructured domains)"**:
   Syntax:

   ```
   UD m e [LOOP]
   ```

   Description:
   Add ligand binding to unpaired sequence motif $m$ (given in IUPAC format, capital letters) with binding energy $e$ in particular loop type(s).
   Example:

   ```
   UD  AAA   -5.0   A
   ```

   The above example applies a binding free energy of $-5kcal/mol$ for a motif AAA that may be present in all loop types.

## 4.3 Plotting

Create Plots of Secondary Structures, Feature Motifs, and Sequence Alignments

### 4.3.1 Producing secondary structure graphs

```
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 rna_plot_type
```

Switch for changing the secondary structure layout algorithm.

Two low-level functions provide direct access to the graph lauyouting algorithms:

```
int simple_xy_coordinates ( short *pair_table,
                            float *X,
                            float *Y)
```

Calculate nucleotide coordinates for secondary structure plot the *Simple way*

```
int naview_xy_coordinates ( short *pair_table,
                            float *X,
                            float *Y)
```

**See also**

> PS_dot.h and naview.h for more detailed descriptions.

### 4.3.2 Producing (colored) dot plots for base pair probabilities

```
int PS_color_dot_plot ( char *string,
                        cpair *pi,
                        char *filename)
```

```
int PS_color_dot_plot_turn (char *seq,
                            cpair *pi,
                            char *filename,
                            int winSize)
```

```
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 PS_dot_plot_turn (char *seq,
                      struct plist *pl,
                      char *filename,
                      int winSize)
```

**See also**

> PS_dot.h for more detailed descriptions.

### 4.3.3 Producing (colored) alignments

```
int PS_color_aln (
            const char *structure,
            const char *filename,
            const char *seqs[],
            const char *names[])
```

Produce PostScript sequence alignment color-annotated by consensus structure.

# Chapter 5

# Basic Data Structures

- Sequence and Structure Data shows the most common types for sequence or structure data
- The 'Fold Compound' is the basic, central container for our implementations of prediction-, evaluation, and other algorithms
- Model Details provides the means to store the different model parameters

## 5.1  Sequence and Structure Data

**See also**

Parsing, converting, comparing secondary structures

## 5.2  The 'Fold Compound'

**See also**

The Fold Compound

## 5.3  Model Details

**See also**

Fine-tuning of the implemented models

# Chapter 6

# API Features

- RNAlib API v3.0
- Callback Functions
- Scripting Language interface(s)

## 6.1  RNAlib API v3.0

### 6.1.1  Introduction

With version 2.2 we introduce the new API that will take over the old one in the future version 3.0. By then, backwards compatibility will be broken, and third party applications using RNAlib need to be ported. This switch of API became necessary, since many new features found their way into the RNAlib where a balance between threadsafety and easy-to-use library functions is hard or even impossible to establish. Furthermore, many old functions of the library are present as slightly modified copies of themself to provide a crude way to overload functions.

Therefore, we introduce the new v3.0 API very early in our development stage such that developers have enough time to migrate to the new functions and interfaces. We also started to provide encapsulation of the RNAlib functions, data structures, typedefs, and macros by prefixing them with *vrna_* and *VRNA_* , respectively. Header files should also be included using the *ViennaRNA/* namespace, e.g.

```
#include <ViennaRNA/fold.h>
```

instead of just using

```
#include <fold.h>
```

as required for RNAlib 1.x and 2.x.

This eases the work for programmers of third party applications that would otherwise need to put much effort into renaming functions and data types in their own implementations if their names appear in our library. Since we still provide backward compatibility up to the last version of RNAlib 2.x, this advantage may be fully exploited only starting from v3.0 which will be released in the future. However, our plan is to provide the possibility for an early switch-off mechanism of the backward compatibility in one of our next releases of ViennaRNA Package 2.x.

### 6.1.2 What are the major changes?

...

### 6.1.3 How to port your program to the new API

...

### 6.1.4 Some Examples using RNAlib API v3.0

Examples on how to use the new v3.0 API can be found in the examples_c_new_API section.

## 6.2 Callback Functions

With the new RNAlib API v3.0 we introduce so-called callback mechanisms for several functions.

### 6.2.1 The purpose of Callback mechanisms

Using callback mechanisms, our library enables users not only to retrieve computed data without the need for parsing complicated data structures, but also allows one to tweak our implementation to do additional tasks without the requirement of a re-implementation of basic algorithms.

Our implementation of the callback mechanisms always follows the same scheme: The user:

- defines a function that complies with the interface we've defined, and

- passes a pointer to said function to our implementations

In addition to the specific arguments of our callback interfaces, virtually all callbacks receive an additional *pass-through-pointer* as their last argument. This enables one to:

- encapsulate data, and

- provide thread-safe operations,

since this pointer is simply passed through by our library functions. It may therefore hold the address of an arbitrary, user-defined data structure.

### 6.2.2 List of available Callbacks

Below, you find an enumeration of the individual callback functions that are available in *RNAlib*.

**Global vrna_callback_free_auxdata (void *data)**

This callback is supposed to free memory occupied by an auxiliary data structure. It will be called when the vrna_fold_compound_t is erased from memory through a call to vrna_fold_compound_free() and will be passed the address of memory previously bound to the vrna_fold_compound_t via vrna_fold_compound_add_auxdata().

**Global vrna_callback_hc_evaluate (int i, int j, int k, int l, unsigned char d, void *data)**

This callback enables one to over-rule default hard constraints in secondary structure decompositions.

**Global vrna_callback_recursion_status (unsigned char status, void *data)**

This function will be called to notify a third-party implementation about the status of a currently ongoing recursion. The purpose of this callback mechanism is to provide users with a simple way to ensure pre- and post conditions for auxiliary mechanisms attached to our implementations.

**Global vrna_callback_sc_backtrack (int i, int j, int k, int l, unsigned char d, void *data)**

This callback enables one to add auxiliary base pairs in the backtracking steps of hairpin- and interior loops.

**Global vrna_callback_sc_energy (int i, int j, int k, int l, unsigned char d, void *data)**

This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure.

**Global vrna_callback_sc_exp_energy (int i, int j, int k, int l, unsigned char d, void *data)**

This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure (Partition function variant, i.e. contributions must be returned as Boltzmann factors).

**Global vrna_callback_ud_energy (vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)**

This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand.

**Global vrna_callback_ud_exp_energy (vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)**

This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand (Partition function variant, i.e. the Boltzmann factors instead of actual free energies).

**Global vrna_callback_ud_exp_production (vrna_fold_compound_t *vc, void *data)**

The production rule for the unstructured domain grammar extension (Partition function variant)

**Global vrna_callback_ud_probs_add (vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, FLT↩ _OR_DBL exp_energy, void *data)**

A callback function to store equilibrium probabilities for the unstructured domain feature

**Global vrna_callback_ud_probs_get (vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, int motif, void *data)**

A callback function to retrieve equilibrium probabilities for the unstructured domain feature

**Global vrna_callback_ud_production (vrna_fold_compound_t *vc, void *data)**

The production rule for the unstructured domain grammar extension

**Global vrna_mfe_window_callback (int start, int end, const char *structure, float en, void *data)**

This function will be called for each hit in a sliding window MFE prediction.

**Global vrna_probs_window_callback (FLT_OR_DBL *pr, int pr_size, int i, int max, unsigned int type, void *data)**

This function will be called for each probability data set in the sliding window probability computation implementation of vrna_probs_window(). The argument *type* specifies the type of probability that is passed to this function.

**Global vrna_subopt_callback (const char *stucture, float energy, void *data)**

This function will be called for each suboptimal secondary structure that is successfully backtraced.

## 6.3 Scripting Language interface(s)

### 6.3.1 Introduction

For an easy integration into scripting languages, we provide an automatically generated interface to the RNAlib C-library, generated with SWIG.

### 6.3.2 Function renaming scheme

The main difference when using a scripting language interface compared to direct calls of RNAlib C functions is, that the prefix 'vrna_' is dropped. For instance, when calling the vrna_fold() function, corresponding calls in perl or python are RNA::fold(), and RNA.fold(), respectively.

Functions that are dedicated to work on specific data structures only, e.g. the vrna_fold_compound_t, are usually not exported at all. Instead, they are attached as object methods of a corresponding class (see Object oriented Interface for data structures for detailed information).

### 6.3.3 Object oriented Interface for data structures

For data structures, typedefs, and enumerations the 'vrna_' prefixes are dropped as well, together with their suffixes '_s', '_t', and '_e', respectively. Furthermore, data structures are usually transformed into classes and relevant functions of the C-library are attached as methods.

### 6.3.4 Examples

Examples on the basic usage of the scripting language interfaces can be found in the scripting_perl_examples and scripting_python_examples section.

### 6.3.5 SWIG generated Wrapper notes

Special notes on how functions, structures, enums, and macro definitions are actually wrapped, can be found below

**Global vrna_aln_conservation_col (const char ∗∗alignment, const vrna_md_t ∗md_p, unsigned int options)**

This function is available in an overloaded form where the last two parameters may be omitted, indicating md = *NULL*, and `options` = VRNA_MEASURE_SHANNON_ENTROPY, respectively.

**Global vrna_aln_conservation_struct (const char ∗∗alignment, const char ∗structure, const vrna_md_↩ t ∗md)**

This function is available in an overloaded form where the last parameter may be omitted, indicating md = *NULL*

**Global vrna_commands_apply (vrna_fold_compound_t ∗vc, vrna_cmd_t ∗commands, unsigned int options)**

This function is attached as method **commands_apply()** to objects of type *fold_compound*

**Global vrna_db_flatten (char ∗structure, unsigned int options)**

This function flattens an input structure string in-place! The second parameter is optional and defaults to VRNA_BRACKETS_DEFAULT.

An overloaded version of this function exists, where an additional second parameter can be passed to specify the target brackets, i.e. the type of matching pair characters all brackets will be flattened to. Therefore, in the scripting language interface this function is a replacement for vrna_db_flatten_to().

**Global vrna_db_flatten_to (char *string, const char target[3], unsigned int options)**

This function is available as an overloaded version of vrna_db_flatten()

**Global vrna_enumerate_necklaces (const unsigned int *type_counts)**

This function is available as global function **enumerate_necklaces()** which accepts lists input, an produces list of lists output.

**Global vrna_eval_circ_consensus_structure (const char **alignment, const char *structure)**

This function is available through an overloadeded version of vrna_eval_circ_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global vrna_eval_circ_consensus_structure_v (const char **alignment, const char *structure, int verbosity_level, FILE *file)**

This function is available through an overloaded version of vrna_eval_circ_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_circ_gquad_consensus_structure (const char **alignment, const char *structure)**

This function is available through an overloadeded version of vrna_eval_circ_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global vrna_eval_circ_gquad_consensus_structure_v (const char **alignment, const char *structure, int verbosity_level, FILE *file)**

This function is available through an overloaded version of vrna_eval_circ_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_circ_gquad_structure (const char *string, const char *structure)**

In the target scripting language, this function serves as a wrapper for vrna_eval_circ_gquad_structure_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_circ_gquad_structure_v (const char *string, const char *structure, int verbosity_level, FILE *file)**

This function is available through an overloaded version of vrna_eval_circ_gquad_structure(). The last two arguments for this function are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_circ_structure (const char *string, const char *structure)**

In the target scripting language, this function serves as a wrapper for vrna_eval_circ_structure_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_V←↩ERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_circ_structure_v (const char *string, const char *structure, int verbosity_level, FILE *file)**

This function is available through an overloaded version of vrna_eval_circ_structure(). The last two arguments for this function are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_consensus_structure_pt_simple (const char **alignment, const short *pt)**

This function is available through an overloadeded version of vrna_eval_structure_pt_simple(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global vrna_eval_consensus_structure_simple (const char **alignment, const char *structure)**

This function is available through an overloadeded version of vrna_eval_structure_simple(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global vrna_eval_consensus_structure_simple_v (const char **alignment, const char *structure, int verbosity_level, FILE *file)**

This function is available through an overloaded version of vrna_eval_structure_simple(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_consensus_structure_simple_verbose (const char **alignment, const char *structure, FILE *file)**

This function is not available. Use vrna_eval_consensus_structure_simple_v() instead!

**Global vrna_eval_covar_structure (vrna_fold_compound_t ∗vc, const char ∗structure)**

This function is attached as method **eval_covar_structure()** to objects of type *fold_compound*

**Global vrna_eval_gquad_consensus_structure (const char ∗∗alignment, const char ∗structure)**

This function is available through an overloadeded version of vrna_eval_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global vrna_eval_gquad_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)**

This function is available through an overloaded version of vrna_eval_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_gquad_structure (const char ∗string, const char ∗structure)**

In the target scripting language, this function serves as a wrapper for vrna_eval_gquad_structure_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_V←ERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_gquad_structure_v (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)**

This function is available through an overloaded version of vrna_eval_gquad_structure(). The last two arguments for this function are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)**

This function is attached as method **eval_hp_loop()** to objects of type *fold_compound*

**Global vrna_eval_int_loop (vrna_fold_compound_t ∗vc, int i, int j, int k, int l)**

This function is attached as method **eval_int_loop()** to objects of type *fold_compound*

**Global vrna_eval_loop_pt (vrna_fold_compound_t ∗vc, int i, const short ∗pt)**

This function is attached as method **eval_loop_pt()** to objects of type *fold_compound*

**Global vrna_eval_move (vrna_fold_compound_t ∗vc, const char ∗structure, int m1, int m2)**

This function is attached as method **eval_move()** to objects of type *fold_compound*

**Global vrna_eval_move_pt (vrna_fold_compound_t ∗vc, short ∗pt, int m1, int m2)**

This function is attached as method **eval_move_pt()** to objects of type *fold_compound*

**Global vrna_eval_structure (vrna_fold_compound_t ∗vc, const char ∗structure)**

This function is attached as method **eval_structure()** to objects of type *fold_compound*

**Global vrna_eval_structure_pt (vrna_fold_compound_t ∗vc, const short ∗pt)**

This function is attached as method **eval_structure_pt()** to objects of type *fold_compound*

**Global vrna_eval_structure_pt_simple (const char ∗string, const short ∗pt)**

In the target scripting language, this function serves as a wrapper for vrna_eval_structure_pt_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_V←ERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_structure_pt_verbose (vrna_fold_compound_t ∗vc, const short ∗pt, FILE ∗file)**

This function is attached as method **eval_structure_pt_verbose()** to objects of type *fold_compound*

**Global vrna_eval_structure_simple (const char ∗string, const char ∗structure)**

In the target scripting language, this function serves as a wrapper for vrna_eval_structure_simple_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_V←ERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_structure_simple_v (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)**

This function is available through an overloaded version of vrna_eval_structure_simple(). The last two arguments for this function are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**Global vrna_eval_structure_simple_verbose (const char ∗string, const char ∗structure, FILE ∗file)**

This function is not available. Use vrna_eval_structure_simple_v() instead!

**Global vrna_eval_structure_verbose (vrna_fold_compound_t ∗vc, const char ∗structure, FILE ∗file)**

This function is attached as method **eval_structure_verbose()** to objects of type *fold_compound*

**Global vrna_exp_params_rescale (vrna_fold_compound_t ∗vc, double ∗mfe)**

This function is attached to vrna_fc_s objects as overloaded **exp_params_rescale()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_rescale(), i.e. default scaling of the partition function. Passing an energy in kcal/mol, e.g. as retrieved by a previous call to the *mfe()* method, instructs all subsequent calls to scale the partition function accordingly.

**Global vrna_exp_params_reset (vrna_fold_compound_t ∗vc, vrna_md_t ∗md_p)**

This function is attached to vrna_fc_s objects as overloaded **exp_params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

**Class vrna_fc_s**

This data structure is wrapped as an object **fold_compound** with several related functions attached as methods.

A new **fold_compound** can be obtained by calling one of its constructors:

- *fold_compound(seq)* – Initialize with a single sequence, or two concatenated sequences separated by an ampersand character '&' (for cofolding)
- *fold_compound(aln)* – Initialize with a sequence alignment *aln* stored as a list of sequences (with gap characters)

The resulting object has a list of attached methods which in most cases directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *type()* – Get the type of the *fold_compound* (See vrna_fc_type_e)
- *length()* – Get the length of the sequence(s) or alignment stored within the *fold_compound*

**Global vrna_file_commands_apply (vrna_fold_compound_t ∗vc, const char ∗filename, unsigned int options)**

This function is attached as method **file_commands_apply()** to objects of type *fold_compound*

**Global vrna_file_msa_detect_format (const char ∗filename, unsigned int options)**

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to VRNA_FILE_FORMAT_MSA_DEFAULT.

**Global vrna_file_msa_read (const char ∗filename, char ∗∗∗names, char ∗∗∗aln, char ∗∗id, char ∗∗structure, unsigned int options)**

In the target scripting language, only the first and last argument, `filename` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
num_seq, names, aln, id, structure = RNA.file_msa_read("msa.stk", RNA.FILE_FORMAT_MSA_STOCKHOLM)
```

After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to VRNA_FILE_FORMAT_MSA_STOCKHOLM.

**Global vrna_file_msa_read_record (FILE ∗fp, char ∗∗∗names, char ∗∗∗aln, char ∗∗id, char ∗∗structure, unsigned int options)**

In the target scripting language, only the first and last argument, `fp` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
f = open('msa.stk', 'r')
num_seq, names, aln, id, structure = RNA.file_msa_read_record(f, RNA.FILE_FORMAT_MSA_STOCKHOLM)
f.close()
```

After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to VRNA_FILE_FORMAT_MSA_STOCKHOLM.

**Global vrna_file_msa_write (const char ∗filename, const char ∗∗names, const char ∗∗aln, const char ∗id, const char ∗structure, const char ∗source, unsigned int options)**

In the target scripting language, this function exists as a set of overloaded versions, where the last four parameters may be omitted. If the `options` parameter is missing the options default to (VRNA_FILE_FORMAT_MSA_STOCKHOLM | VRNA_FILE_FORMAT_MSA_APPEND).

**Global vrna_hc_add_from_db (vrna_fold_compound_t ∗vc, const char ∗constraint, unsigned int options)**

This function is attached as method **hc_add_from_db()** to objects of type *fold_compound*

**Global vrna_hc_init (vrna_fold_compound_t ∗vc)**

This function is attached as method **hc_init()** to objects of type *fold_compound*

**Class vrna_md_s**

This data structure is wrapped as an object **md** with multiple related functions attached as methods.

A new set of default parameters can be obtained by calling the constructure of **md:**

- *md()* – Initialize with default settings

The resulting object has a list of attached methods which directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *reset()* – vrna_md_set_default()
- *set_from_globals()* – set_model_details()
- *option_string()* – vrna_md_option_string()

Note, that default parameters can be modified by directly setting any of the following global variables. Internally, getting/setting default parameters using their global variable representative translates into calls of the following functions, therefore these wrappers for these functions do not exist in the scripting language interface(s):

| global variable | *C* getter | *C* setter |
|---|---|---|
| temperature | vrna_md_defaults_temperature_get() | vrna_md_defaults_temperature() |
| dangles | vrna_md_defaults_dangles_get() | vrna_md_defaults_dangles() |
| betaScale | vrna_md_defaults_betaScale_get() | vrna_md_defaults_betaScale() |
| tetra_loop | this is an alias of *special_hp* | |
| special_hp | vrna_md_defaults_special_hp_get() | vrna_md_defaults_special_hp() |
| noLonelyPairs | this is an alias of *noLP* | |
| noLP | vrna_md_defaults_noLP_get() | vrna_md_defaults_noLP() |

| global variable | *C* getter | *C* setter |
|---|---|---|
| noGU | vrna_md_defaults_noGU_get() | vrna_md_defaults_noGU() |
| no_closingGU | this is an alias of *noGUclosure* | |
| noGUclosure | vrna_md_defaults_noGUclosure_get() | vrna_md_defaults_noGUclosure() |
| logML | vrna_md_defaults_logML_get() | vrna_md_defaults_logML() |
| circ | vrna_md_defaults_circ_get() | vrna_md_defaults_circ() |
| gquad | vrna_md_defaults_gquad_get() | vrna_md_defaults_gquad() |
| uniq_ML | vrna_md_defaults_uniq_ML_get() | vrna_md_defaults_uniq_ML() |
| energy_set | vrna_md_defaults_energy_set_get() | vrna_md_defaults_energy_set() |
| backtrack | vrna_md_defaults_backtrack_get() | vrna_md_defaults_backtrack() |
| backtrack_type | vrna_md_defaults_backtrack_type_get() | vrna_md_defaults_backtrack_type() |
| do_backtrack | this is an alias of *compute_bpp* | |
| compute_bpp | vrna_md_defaults_compute_bpp_get() | vrna_md_defaults_compute_bpp() |
| max_bp_span | vrna_md_defaults_max_bp_span_get() | vrna_md_defaults_max_bp_span() |
| min_loop_size | vrna_md_defaults_min_loop_size_get() | vrna_md_defaults_min_loop_size() |
| window_size | vrna_md_defaults_window_size_get() | vrna_md_defaults_window_size() |
| oldAliEn | vrna_md_defaults_oldAliEn_get() | vrna_md_defaults_oldAliEn() |
| ribo | vrna_md_defaults_ribo_get() | vrna_md_defaults_ribo() |
| cv_fact | vrna_md_defaults_cv_fact_get() | vrna_md_defaults_cv_fact() |
| nc_fact | vrna_md_defaults_nc_fact_get() | vrna_md_defaults_nc_fact() |
| sfact | vrna_md_defaults_sfact_get() | vrna_md_defaults_sfact() |

**Global vrna_mean_bp_distance (vrna_fold_compound_t ∗vc)**

This function is attached as method **mean_bp_distance()** to objects of type *fold_compound*

**Global vrna_mfe (vrna_fold_compound_t ∗vc, char ∗structure)**

This function is attached as method **mfe()** to objects of type *fold_compound*

**Global vrna_mfe_dimer (vrna_fold_compound_t ∗vc, char ∗structure)**

This function is attached as method **mfe_dimer()** to objects of type *fold_compound*

**Global vrna_mfe_window (vrna_fold_compound_t ∗vc, FILE ∗file)**

This function is attached as method **mfe_window()** to objects of type *fold_compound*

**Global vrna_neighbors (vrna_fold_compound_t ∗vc, const short ∗pt, unsigned int options)**

This function is attached as an overloaded method *neighbors()* to objects of type *fold_compound*. The optional parameter `options` defaults to VRNA_MOVESET_DEFAULT if it is omitted.

**Global vrna_params_reset (vrna_fold_compound_t ∗vc, vrna_md_t ∗md_p)**

This function is attached to vrna_fc_s objects as overloaded **params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

**Global vrna_params_subst (vrna_fold_compound_t ∗vc, vrna_param_t ∗par)**

This function is attached to vrna_fc_s objects as **params_subst()** method.

**Global vrna_path (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int steps, unsigned int options)**

This function is attached as an overloaded method *path()* to objects of type *fold_compound*. The optional parameter `options` defaults to VRNA_PATH_DEFAULT if it is omitted.

**Global vrna_path_findpath (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width)**

This function is attached as an overloaded method *path_findpath()* to objects of type *fold_compound*. The optional parameter width defaults to 1 if it is omitted.

**Global vrna_path_findpath_saddle (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width)**

This function is attached as an overloaded method *path_findpath_saddle()* to objects of type *fold_compound*. The optional parameter width defaults to 1 if it is omitted.

**Global vrna_path_findpath_saddle_ub (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width, int maxE)**

This function is attached as an overloaded method *path_findpath_saddle()* to objects of type *fold_compound*. The optional parameter width defaults to 1 if it is omitted, while the optional parameter maxE defaults to INF. In case the function did not find a path with $E_{saddle} < E_{max}$ the function returns a *NULL* object, i.e. *undef* for Perl and *None* for Python.

**Global vrna_path_findpath_ub (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width, int maxE)**

This function is attached as an overloaded method *path_findpath()* to objects of type *fold_compound*. The optional parameter width defaults to 1 if it is omitted, while the optional parameter maxE defaults to INF. In case the function did not find a path with $E_{saddle} < E_{max}$ the function returns an empty list.

**Global vrna_path_gradient (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int options)**

This function is attached as an overloaded method *path_gradient()* to objects of type *fold_compound*. The optional parameter options defaults to VRNA_PATH_DEFAULT if it is omitted.

**Global vrna_path_random (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int steps, unsigned int options)**

This function is attached as an overloaded method *path_gradient()* to objects of type *fold_compound*. The optional parameter options defaults to VRNA_PATH_DEFAULT if it is omitted.

**Global vrna_pbacktrack (vrna_fold_compound_t ∗vc)**

This function is attached as overloaded method **pbacktrack()** to objects of type *fold_compound* that accepts an optional length argument. Hence, it serves as a replacement for vrna_pbacktrack().

**Global vrna_pbacktrack5 (vrna_fold_compound_t ∗vc, int length)**

This function is attached as overloaded method **pbacktrack()** to objects of type *fold_compound*

**Global vrna_pf (vrna_fold_compound_t ∗vc, char ∗structure)**

This function is attached as method **pf()** to objects of type *fold_compound*

**Global vrna_pf_dimer (vrna_fold_compound_t ∗vc, char ∗structure)**

This function is attached as method **pf_dimer()** to objects of type *fold_compound*

**Global vrna_sc_add_bp (vrna_fold_compound_t ∗vc, int i, int j, FLT_OR_DBL energy, unsigned int options)**

This function is attached as an overloaded method **sc_add_bp()** to objects of type *fold_compound*. The method either takes arguments for a single base pair (i,j) with the corresponding energy value:

```
fold_compound.sc_add_bp(i, j, energy, options)
```

or an entire 2-dimensional matrix with dimensions n x n that stores free energy contributions for any base pair (i,j) with $1 \le i < j \le n$:

```
fold_compound.sc_add_bp(matrix, options)
```

In both variants, the options argument is optional can may be omitted.

**Global vrna_sc_add_bt (vrna_fold_compound_t ∗vc, vrna_callback_sc_backtrack ∗f)**

This function is attached as method **sc_add_bt()** to objects of type *fold_compound*

**Global vrna_sc_add_data (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗free_data)**

This function is attached as method **sc_add_data()** to objects of type *fold_compound*

**Global vrna_sc_add_exp_f (vrna_fold_compound_t ∗vc, vrna_callback_sc_exp_energy ∗exp_f)**

This function is attached as method **sc_add_exp_f()** to objects of type *fold_compound*

**Global vrna_sc_add_f (vrna_fold_compound_t ∗vc, vrna_callback_sc_energy ∗f)**

This function is attached as method **sc_add_f()** to objects of type *fold_compound*

**Global vrna_sc_add_hi_motif (vrna_fold_compound_t ∗vc, const char ∗seq, const char ∗structure, FLT_↩ OR_DBL energy, unsigned int options)**

This function is attached as method **sc_add_hi_motif()** to objects of type *fold_compound*

**Global vrna_sc_add_SHAPE_deigan (vrna_fold_compound_t ∗vc, const double ∗reactivities, double m, double b, unsigned int options)**

This function is attached as method **sc_add_SHAPE_deigan()** to objects of type *fold_compound*

**Global vrna_sc_add_SHAPE_deigan_ali (vrna_fold_compound_t ∗vc, const char ∗∗shape_files, const int ∗shape_file_association, double m, double b, unsigned int options)**

This function is attached as method **sc_add_SHAPE_deigan_ali()** to objects of type *fold_compound*

**Global vrna_sc_add_SHAPE_zarringhalam (vrna_fold_compound_t ∗vc, const double ∗reactivities, double b, double default_value, const char ∗shape_conversion, unsigned int options)**

This function is attached as method **sc_add_SHAPE_zarringhalam()** to objects of type *fold_compound*

**Global vrna_sc_add_up (vrna_fold_compound_t ∗vc, int i, FLT_OR_DBL energy, unsigned int options)**

This function is attached as an overloaded method **sc_add_up()** to objects of type *fold_compound*. The method either takes arguments for a single nucleotide $i$ with the corresponding energy value:

```
fold_compound.sc_add_up(i, energy, options)
```

or an entire vector that stores free energy contributions for each nucleotide $i$ with $1 \leq i \leq n$:

```
fold_compound.sc_add_bp(vector, options)
```

In both variants, the `options` argument is optional can may be omitted.

**Global vrna_sc_init (vrna_fold_compound_t ∗vc)**

This function is attached as method **sc_init()** to objects of type *fold_compound*

**Global vrna_sc_remove (vrna_fold_compound_t ∗vc)**

This function is attached as method **sc_remove()** to objects of type *fold_compound*

**Global vrna_sc_set_bp (vrna_fold_compound_t ∗vc, const FLT_OR_DBL ∗∗constraints, unsigned int options)**

This function is attached as method **sc_set_bp()** to objects of type *fold_compound*

**Global vrna_sc_set_up (vrna_fold_compound_t ∗vc, const FLT_OR_DBL ∗constraints, unsigned int options)**

This function is attached as method **sc_set_up()** to objects of type *fold_compound*

**Global vrna_subopt (vrna_fold_compound_t ∗vc, int delta, int sorted, FILE ∗fp)**

This function is attached as method **subopt()** to objects of type *fold_compound*

**Global vrna_subopt_cb (vrna_fold_compound_t ∗vc, int delta, vrna_subopt_callback ∗cb, void ∗data)**

This function is attached as method **subopt_cb()** to objects of type *fold_compound*

**Global vrna_subopt_zuker (vrna_fold_compound_t ∗vc)**

This function is attached as method **subopt_zuker()** to objects of type *fold_compound*

**Global vrna_ud_add_motif (vrna_fold_compound_t ∗vc, const char ∗motif, double motif_en, unsigned int loop_type)**

This function is attached as method **ud_add_motif()** to objects of type *fold_compound*

**Global vrna_ud_remove (vrna_fold_compound_t ∗vc)**

This function is attached as method **ud_remove()** to objects of type *fold_compound*

**Global vrna_ud_set_data (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗free_cb)**

This function is attached as method **ud_set_data()** to objects of type *fold_compound*

**Global** [vrna_ud_set_exp_prod_rule_cb](#) **(vrna_fold_compound_t** ∗**vc, vrna_callback_ud_exp_production** ∗**pre_cb, vrna_callback_ud_exp_energy** ∗**exp_e_cb)**

This function is attached as method **ud_set_exp_prod_rule_cb()** to objects of type *fold_compound*

**Global** [vrna_ud_set_prob_cb](#) **(vrna_fold_compound_t** ∗**vc, vrna_callback_ud_probs_add** ∗**setter, vrna_↩ callback_ud_probs_get** ∗**getter)**

This function is attached as method **ud_set_prob_cb()** to objects of type *fold_compound*

**Global** [vrna_ud_set_prod_rule_cb](#) **(vrna_fold_compound_t** ∗**vc, vrna_callback_ud_production** ∗**pre_cb, vrna_callback_ud_energy** ∗**e_cb)**

This function is attached as method **ud_set_prod_rule_cb()** to objects of type *fold_compound*

**Chapter 7**

# Additional Utilities

# Chapter 8

# Examples

- C Examples
- Perl5 Examples
- Python Examples

## 8.1 C Examples

### 8.1.1 Hello World Examples

**helloworld_mfe.c**

The following is an example showing the minimal requirements to compute the Minimum Free Energy (MFE) and corresponding secondary structure of an RNA sequence

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* The RNA sequence */
  char  *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

  /* allocate memory for MFE structure (length + 1) */
  char  *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* predict Minmum Free Energy and corresponding secondary structure */
  float mfe = vrna_fold(seq, structure);

  /* print sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

  /* cleanup memory */
  free(structure);

  return 0;
}
```

**See also**

> examples/helloworld_mfe.c in the source code tarball

**helloworld_mfe_comparative.c**

Instead of using a single sequence as done above, this example predicts a consensus structure for a multiple sequence alignment

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/alifold.h>
#include <ViennaRNA/aln_util.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* The RNA sequence alignment */
  const char  *sequences[] = {
    "CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUACCCAUCGGUGCAGUGCGGGU",
    NULL /* indicates end of alignment */
  };

  /* compute the consensus sequence */
  char        *cons = consensus(sequences);

  /* allocate memory for MFE consensus structure (length + 1) */
  char        *structure = (char *)vrna_alloc(sizeof(char) * (strlen(sequences[0]) + 1));

  /* predict Minmum Free Energy and corresponding secondary structure */
  float       mfe = vrna_alifold(sequences, structure);

  /* print consensus sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", cons, structure, mfe);

  /* cleanup memory */
  free(cons);
  free(structure);

  return 0;
}
```

**See also**

> examples/helloworld_mfe_comparative.c in the source code tarball

**helloworld_probabilities.c**

This example shows how to compute the partition function and base pair probabilities with minimal implementation effort.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* The RNA sequence */
  char        *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

  /* allocate memory for pairing propensity string (length + 1) */
  char        *propensity = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* pointers for storing and navigating through base pair probabilities */
  vrna_ep_t *ptr, *pair_probabilities = NULL;

  float       en = vrna_pf_fold(seq, propensity, &pair_probabilities);
```

```c
/* print sequence, pairing propensity string and ensemble free energy */
printf("%s\n%s [ %6.2f ]\n", seq, propensity, en);

/* print all base pairs with probability above 50% */
for (ptr = pair_probabilities; ptr->i != 0; ptr++)
  if (ptr->p > 0.5)
    printf("p(%d, %d) = %g\n", ptr->i, ptr->j, ptr->p);

/* cleanup memory */
free(pair_probabilities);
free(propensity);

return 0;
}
```

**See also**

> examples/helloworld_probabilities.c in the source code tarball

### 8.1.2   First Steps with the Fold Compound

**fold_compound_mfe.c**

Instead of calling the simple MFE folding interface vrna_fold(), this example shows how to first create a vrna_fold_compound_t container with the RNA sequence to finally compute the MFE using this container. This is especially useful if non-default model settings are applied or the dynamic programming (DP) matrices of the MFE prediction are required for post-processing operations, or other tasks on the same sequence will be performed.

```c
#include <stdlib.h>
#include <stdio.h>

#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/mfe.h>
#include <ViennaRNA/string_utils.h>
#include <ViennaRNA/utils.h>


int
main()
{
  /* initialize random number generator */
  vrna_init_rand();

  /* Generate a random sequence of 50 nucleotides */
  char                *seq = vrna_random_string(50, "ACGU");

  /* Create a fold compound for the sequence */
  vrna_fold_compound_t  *fc = vrna_fold_compound(seq, NULL,
      VRNA_OPTION_DEFAULT);

  /* allocate memory for MFE structure (length + 1) */
  char                *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* predict Minmum Free Energy and corresponding secondary structure */
  float               mfe = vrna_mfe(fc, structure);

  /* print sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

  /* cleanup memory */
  free(seq);
  free(structure);
  vrna_fold_compound_free(fc);

  return 0;
}
```

**See also**

> examples/fold_compound_mfe.c in the source code tarball

**fold_compound_md.c**

In the following, we change the model settings (model details) to a temperature of 25 Degree Celcius, and activate G-Quadruplex precition.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/mfe.h>
#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/model.h>
#include <ViennaRNA/string_utils.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* initialize random number generator */
  vrna_init_rand();

  /* Generate a random sequence of 50 nucleotides */
  char        *seq = vrna_random_string(50, "ACGU");

  /* allocate memory for MFE structure (length + 1) */
  char        *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* create a new model details structure to store the Model Settings */
  vrna_md_t md;

  /* ALWAYS set default model settings first! */
  vrna_md_set_default(&md);

  /* change temperature and activate G-Quadruplex prediction */
  md.temperature  = 25.0; /* 25 Deg Celcius */
  md.gquad        = 1;     /* Turn-on G-Quadruples support */

  /* create a fold compound */
  vrna_fold_compound_t  *fc = vrna_fold_compound(seq, &md,
      VRNA_OPTION_DEFAULT);

  /* predict Minmum Free Energy and corresponding secondary structure */
  float           mfe = vrna_mfe(fc, structure);

  /* print sequence, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

  /* cleanup memory */
  free(structure);
  vrna_fold_compound_free(fc);

  return 0;
}
```

**See also**

> examples/fold_compound_md.c in the source code tarball

### 8.1.3 Writing Callback Functions

**callback_subopt.c**

Here is a basic example how to use the callback mechanism in vrna_subopt_cb(). It simply defines a callback function (see interface definition for vrna_subopt_callback) that prints the result and increases a counter variable.

```c
#include <stdlib.h>
#include <stdio.h>

#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/subopt.h>
#include <ViennaRNA/string_utils.h>
```

```c
#include <ViennaRNA/utils.h>


void
subopt_callback(const char  *structure,
                float       energy,
                void        *data)
{
  /* simply print the result and increase the counter variable by 1 */
  if (structure)
    printf("%d.\t%s\t%6.2f\n", (*((int *)data))++, structure, energy);
}


int
main()
{
  /* initialize random number generator */
  vrna_init_rand();

  /* Generate a random sequence of 50 nucleotides */
  char                *seq = vrna_random_string(50, "ACGU");

  /* Create a fold compound for the sequence */
  vrna_fold_compound_t  *fc = vrna_fold_compound(seq, NULL,
      VRNA_OPTION_DEFAULT);

  int                 counter = 0;

  /*
   *  call subopt to enumerate all secondary structures in an energy band of
   *  5 kcal/mol of the MFE and pass it the address of the callback and counter
   *  variable
   */
  vrna_subopt_cb(fc, 500, &subopt_callback, (void *)&counter);

  /* cleanup memory */
  free(seq);
  vrna_fold_compound_free(fc);

  return 0;
}
```

**See also**

> examples/callback_subopt.c in the source code tarball

### 8.1.4 Application of Soft Constraints

**soft_constraints_up.c**

In this example, a random RNA sequence is generated to predict its MFE under the constraint that a particular nucleotide receives an additional bonus energy if it remains unpaired.

```c
#include <stdlib.h>
#include <stdio.h>

#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/mfe.h>
#include <ViennaRNA/string_utils.h>
#include <ViennaRNA/constraints_soft.h>
#include <ViennaRNA/utils.h>

int
main()
{
  /* initialize random number generator */
  vrna_init_rand();

  /* Generate a random sequence of 50 nucleotides */
  char                *seq = vrna_random_string(50, "ACGU");

  /* Create a fold compound for the sequence */
  vrna_fold_compound_t  *fc = vrna_fold_compound(seq, NULL,
      VRNA_OPTION_DEFAULT);
```

```
  /* Add soft constraint of -1.7 kcal/mol to nucleotide 5 whenever it appears in an unpaired context */
  vrna_sc_add_up(fc, 5, -1.7, VRNA_OPTION_DEFAULT);

  /* allocate memory for MFE structure (length + 1) */
  char  *structure = (char *)vrna_alloc(sizeof(char) * 51);

  /* predict Minmum Free Energy and corresponding secondary structure */
  float mfe = vrna_mfe(fc, structure);

  /* print seqeunce, structure and MFE */
  printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

  /* cleanup memory */
  free(seq);
  free(structure);
  vrna_fold_compound_free(fc);

  return 0;
}
```

**See also**

>    examples/soft_constraints_up.c in the source code tarball

### 8.1.5    Other Examples

**example1.c**

A more extensive example including MFE, Partition Function, and Centroid structure prediction.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <ViennaRNA/data_structures.h>
#include <ViennaRNA/params.h>
#include <ViennaRNA/utils.h>
#include <ViennaRNA/eval.h>
#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>

int
main(int  argc,
     char *argv[])
{
  char                *seq =
    "AGACGACAAGGUUGAAUCGCACCCACAGUCUAUGAGUCGGUGACAACAUUACGAAAGGCUGUAAAAUCAAUUAUUCACCACAGGGGGCCCCCGUGUCUAG";
  char                *mfe_structure = vrna_alloc(sizeof(char) * (strlen(seq) + 1));
  char                *prob_string   = vrna_alloc(sizeof(char) * (strlen(seq) + 1));

  /* get a vrna_fold_compound with default settings */
  vrna_fold_compound_t  *vc = vrna_fold_compound(seq, NULL,
      VRNA_OPTION_DEFAULT);

  /* call MFE function */
  double              mfe = (double)vrna_mfe(vc, mfe_structure);

  printf("%s\n%s (%6.2f)\n", seq, mfe_structure, mfe);

  /* rescale parameters for Boltzmann factors */
  vrna_exp_params_rescale(vc, &mfe);

  /* call PF function */
  FLT_OR_DBL en = vrna_pf(vc, prob_string);

  /* print probability string and free energy of ensemble */
  printf("%s (%6.2f)\n", prob_string, en);

  /* compute centroid structure */
  double  dist;
  char    *cent = vrna_centroid(vc, &dist);

  /* print centroid structure, its free energy and mean distance to the ensemble */
  printf("%s (%6.2f d=%6.2f)\n", cent, vrna_eval_structure(vc, cent), dist);
```

```
    /* free centroid structure */
    free(cent);

    /* free pseudo dot-bracket probability string */
    free(prob_string);

    /* free mfe structure */
    free(mfe_structure);

    /* free memory occupied by vrna_fold_compound */
    vrna_fold_compound_free(vc);

    return EXIT_SUCCESS;
}
```

**See also**

> examples/example1.c in the source code tarball

## 8.1.6 Deprecated Examples

```
#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);
}
```

**See also**

> `examples/example_old.c` in the source code tarball

## 8.2   Perl5 Examples

## Hello World Examples

### Using the flat interface

- MFE prediction

```perl
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

# compute minimum free energy (MFE) and corresponding structure
my ($ss, $mfe) = RNA::fold($seq);

# print output
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

### Using the object oriented interface

- MFE prediction

```perl
#!/usr/bin/perl

use warnings;
use strict;

use RNA;

my $seq1 = "CGCAGGGAUACCCGCG";

# create new fold_compound object
my $fc = new RNA::fold_compound($seq1);

# compute minimum free energy (mfe) and corresponding structure
my ($ss, $mfe) = $fc->mfe();

# print output
printf "%s [ %6.2f ]\n", $ss, $mfe;
```

## Changing the Model Settings

### Using the flat interface

- MFE prediction at different temperature and dangle model

```
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

# create a new model details structure
my $md = new RNA::md();

# change temperature and dangle model
$md->{temperature} = 20.0; # 20 Deg Celcius
$md->{dangles}     = 1;    # Dangle Model 1

# create a fold compound
my $fc = new RNA::fold_compound($seq, $md);

# predict Minmum Free Energy and corresponding secondary structure
my ($ss, $mfe) = $fc->mfe();

# print sequence, structure and MFE
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

### Using the object oriented interface

- MFE prediction at different temperature and dangle model

## 8.3 Python Examples

### MFE Prediction (flat interface)

```
import RNA

# The RNA sequence
seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA"

# compute minimum free energy (MFE) and corresponding structure
(ss, mfe) = RNA.fold(seq)

# print output
print "%s\n%s [ %6.2f ]" % (seq, ss, mfe)
```

### MFE Prediction (object oriented interface)

```
import RNA;

sequence = "CGCAGGGAUACCCGCG"

# create new fold_compound object
fc = RNA.fold_compound(sequence)

# compute minimum free energy (mfe) and corresponding structure
(ss, mfe) = fc.mfe()

# print output
print "%s [ %6.2f ]" % (ss, mfe)
```

```python
import RNA

sequence = "GGGGAAAACCCC"

# Set global switch for unique ML decomposition
RNA.cvar.uniq_ML = 1

subopt_data = { 'counter' : 1, 'sequence' : sequence }

# Print a subopt result as FASTA record
def print_subopt_result(structure, energy, data):
    if not structure == None:
        print ">subopt %d" % data['counter']
        print "%s" % data['sequence']
        print "%s [%6.2f]" % (structure, energy)
        # increase structure counter
        data['counter'] = data['counter'] + 1

# Create a 'fold_compound' for our sequence
a = RNA.fold_compound(sequence)

# Enumerate all structures 500 dacal/mol = 5 kcal/mol arround
# the MFE and print each structure using the function above
a.subopt_cb(500, print_subopt_result, subopt_data);
```

```python
import RNA

seq1 = "CUCGUCGCCUUAAUCCAGUGCGGGCGCUAGACAUCUAGUUAUCGCCGCAA"

# Turn-off dangles globally
RNA.cvar.dangles = 0

# Data structure that will be passed to our MaximumMatching() callback with two components:
# 1. a 'dummy' fold_compound to evaluate loop energies w/o constraints, 2. a fresh set of energy parameters
mm_data = { 'dummy': RNA.fold_compound(seq1), 'params': RNA.param() }

# Nearest Neighbor Parameter reversal functions
revert_NN = {
    RNA.DECOMP_PAIR_HP:        lambda i, j, k, l, f, p: - f.eval_hp_loop(i, j) - 100,
    RNA.DECOMP_PAIR_IL:        lambda i, j, k, l, f, p: - f.eval_int_loop(i, j, k, l) - 100,
    RNA.DECOMP_PAIR_ML:        lambda i, j, k, l, f, p: - p.MLclosing - p.MLintern[0] - (j - i - k + l - 2)
        * p.MLbase - 100,
    RNA.DECOMP_ML_ML_STEM:     lambda i, j, k, l, f, p: - p.MLintern[0] - (l - k - 1) * p.MLbase,
    RNA.DECOMP_ML_STEM:        lambda i, j, k, l, f, p: - p.MLintern[0] - (j - i - k + l) * p.MLbase,
    RNA.DECOMP_ML_ML:          lambda i, j, k, l, f, p: - (j - i - k + l) * p.MLbase,
    RNA.DECOMP_ML_UP:          lambda i, j, k, l, f, p: - (j - i + 1) * p.MLbase,
    RNA.DECOMP_EXT_STEM:       lambda i, j, k, l, f, p: - f.E_ext_loop(k, l),
    RNA.DECOMP_EXT_STEM_EXT:   lambda i, j, k, l, f, p: - f.E_ext_loop(i, k),
    RNA.DECOMP_EXT_EXT_STEM:   lambda i, j, k, l, f, p: - f.E_ext_loop(l, j),
    RNA.DECOMP_EXT_EXT_STEM1:  lambda i, j, k, l, f, p: - f.E_ext_loop(l, j-1),
            }

# Maximum Matching callback function (will be called by RNAlib in each decomposition step)
def MaximumMatching(i, j, k, l, d, data):
    return revert_NN[d](i, j, k, l, data['dummy'], data['params'])

# Create a 'fold_compound' for our sequence
fc = RNA.fold_compound(seq1)

# Add maximum matching soft-constraints
fc.sc_add_f(MaximumMatching)
fc.sc_add_data(mm_data, None)

# Call MFE algorithm
(s, mm) = fc.mfe()

# print result
print "%s\n%s (MM: %d)\n" %  (seq1, s, -mm)
```

# Chapter 9

# Changelog

v2.4.6

- Stabilize rounding of free energy output in RNAalifold

- API: Fix potential rounding errors for comparative free energies in eval.c and mfe.c

- API: Fix regression in exterior loop dangling end contributions for comparative base pair probabilities and Boltzmann sampling (introduced with v2.4.4)

- API: Fix regression with hard constrained base pairs for comparative structure prediction (introduced with v2.4.4)

- TESTS: Add basic tests for RNAalifold executable

- TESTS: Ignore 'frequency of MFE structure' in RNAcofold partition function checks

v2.4.5

- Allow for arbitrary number of input files in RNAfold

- Allow for parallel processing of input data in RNAfold (UNIX only, no Windows support yet)

- Add SHAPE reactivity support through commandline options for RNAplfold

- Fix unstructured domain motif detection in MFE, centroid, and MEA structures computed by RNAfold

- Limit allowed set of commands in command file for RNAcofold to hard and soft constraints

- API: Add functions to compute equilibrium probability of particular secondary structures

- API: Add dynamic string stream data type and associated functions

- API: Add priority-queue like data structure with unordered fill capability and ordered output callback execution

- API: Add functions to detect unstructured domain motifs in MFE, centroid, and MEA structures

- API: Fix bug in sliding-window partition function computation with SHAPE reactivity and Deigan et al. conversion method

- API: Fix application of '$<$' and '$>$' constraint symbols in dot-bracket provided constraints (was broken since v2.4.2)

- API: Fix MEA structure computation in the presence of unstructured domains

- API: Stabilize order of probability entries in EPS dot-plot files

- Fix compiler warnings on wrong type of printf() in naview.c

- Define VRNA_VERSION macro as string literal and add macros for major, minor, and patch numbers

- Stabilize parallel make of Mac OS X installer

- Add energy parameter set from Langdon et al. 2018

- Add autoconf checks for POSIX threads compiler/linker support

- SWIG: Fix 'next' is a perl keyword warnings for Perl5 wrapper

- SWIG: Catch errors and throw execptions whenever scripting language provided callback functions are not applicable or fail

- SWIG: Add keyword arguments and autodoc feature for Python/Python3 wrappers

v2.4.4

- Change verbose output for soft-constraints derived ligand binding motifs in RNAfold

- Allow for lowercase letters in ct2db input

- Fix bug in interior-like G-Quadruplex MFE computation for single sequences

- Fix autoconf switch to enable deprecation warnings

- Fix bug in eval_int_loop() that prevented propagation of energy evaluation for loops with nick in strands

- Fix several bugs for SHAPE reactivity related comparative partition function computations

- Fix annotation of PostScript output for soft-constraint derived ligand binding motifs in RNAfold

- Fix constraint indices for multibranch loops in unpaired probability computations of LPfold.c

- Fix dangling end contributions in comparative partition function for exterior loops

- API: Add simplified interface for vrna_pf_dimer()

- API: Move concentraton dependent implementation for co-folding to separate compile unit

- API: Add new API functions for exterior loop evaluations

- API: Add simplified interfaces for energy evaluation with G-Quadruplexes and circular RNAs

- API: Add findpath functions that allow for specification of an upper bound for the saddle point

- Add configure-time linker check for Python3 interface

- Add automatic CPP suggestions for deprecated function substitutes

- Major restucturing and constraints feature additions in loop type dependent energy evaluation functions

- Major restructuring in MFE implementations

- Major restructuring in PF implementations

- Minor fixes in Boltzmann sampling implementation

- SWIG: Fix wrappers for findpath() implementation

- SWIG: Add tons of energy evaluation wrappers

- SWIG: Fix configure-time check of Perl5 interface build capabilities

- SWIG: Wrap functions from walk.c and neighbor.c

- DOC: Add some missing references to manpages of executable programs

- REFMAN: Heavy re-ordering of the RNAlib reference manual

v2.4.3

- Fix handling of dangling end contribution at sequence boundaries for sliding window base pair probability computations

- Fix handling of base pair hard constraints in sliding-window implementations

- Fix sliding-window pair probability computations with multibranch-loop unpaired constraints

- Fix sliding-window non-specific base pair hard constraint implementation

- Fix probability computation for stochastic backtracking in RNAsubopt --stochBT_en output

- Fix regression in comparative structure prediction for circular RNAs

- Fix LDFLAGS for scripting language interfaces in corresponding Makefiles

- Stabilize partition function scaling by always using sfact scaling factor from model details

- Add --pf_scale commandling parameter to RNAplfold

- Add constraint framework for single sequence circular RNA structure prediction

- Add RNAfold test suite to check for working implementation of constraints for circular RNAs

- Add a brief contribution guideline CONTRIBUTING.md

- Prevent RNAplfold from creating inf/-inf output when solution set is empty with particular hard constraints

- Include RNAforester v2.0.1

v2.4.2

- Fix G-Quadruplex energy corrections in comparative structure energy evaluations

- Fix discrepancy in comparative exterior loop dangling end contribution of eval vs. MFE predictions

- Fix regression in RNAup unstructuredness and interaction energy computations

- Fix sequence length confusions when FASTA input contains carriage returns

- Fix build problems of RNAlocmin with older compilers

- Fix sliding-window hard constraints where single nucleotides are prohibited from pairing

- Fix dot-bracket output string length in sliding-window MFE with G-Quadruplexes

- Fix unpaired probability computations for separate individual loop types in LPfold.c

- Fix bad memory access in RNAsubopt with dot-bracket constraint

- Add full WUSS support for --SS_cons constraint option in RNAalifold

- Add commandline option to RNALalifold that enables splitting of energy contributions into separate parts

- Add missing hard constraint cases to sliding-window partition function implementation

- Add CSV output option to RNAcofold

- Use the same model details for SCI computations in RNAalifold

- Abort computations in vrna_eval_structure_v() if structure has unexpected length

- Use original MSA in all output generated by RNAalifold and RNALalifold

- API: Add new functions to convert dot-bracket like structure annotations

- API: Add various new utility functions for alignment handling and comparative structure predictions

- API: Add function vrna_strsplit() to split string into tokens

- API: Do not convert sequences of input MSA to uppercase letters in vrna_file_msa_read_record()

- API: Rename vrna_annotate_bp_covar() and vrna_annotate_pr_covar()

- API: Add new noLP neighbor generation

- SWIG: Add wrapper for functions in file_utils_msa.h

- SWIG: Add wrappers for vrna_pbacktrack() and vrna_pbacktrack5()

- SWIG: Add vrna_db_to_element_string() to scripting language interface

- REFMAN: Fix formula to image conversion in HTML output

v2.4.1

- Fix memory leak in fold_compound methods of SWIG interface

- Fix memory leaks in double ∗∗ returning functions of SWIG Perl5 interface

- Fix memory leak in vrna_ep_t to-string() function of SWIG interface

- Regression: Fix reverting pf_scale to defaults after vrna_exp_params_rescale()

- Regression: Fix homo-dimer partition function computation in RNAcofold

- Add unit tests for RNAcofold executable

- Add SHAPE reactivity support to RNAcofold

- Add SHAPE reactivity support to RNALalifold

v2.4.0

- Bump libsvm to version 3.22

- Print G-Quadruplex corrections in verbose mode of RNAeval

- Change behavior of RNAfold --outfile option to something more predictable

- Unify max_bp_span usage among sliding window prediction algorithms: RNAplfold, RNALfold, and RNA↩ Lalifold now consider any base pair (i,j) with (j - i + 1) <= max_bp_span

- Add SHAPE reactivity data support to RNALfold

- Add commands-file support for RNALfold, RNAplfold (hard/soft constraints)

- Add RNAlocmin - Calculate local minima from structures via gradient walks

- Add RNA Bioinformatics tutorial (PDF version)

- Add hard constraints to sliding-window MFE implementations (RNALfold, RNALalifold)

- Add hard constraints to sliding-window PF implementations (RNAplfold)

- Add soft constraints to sliding-window MFE implementation for single sequences (RNALfold)

- Add soft constraints to sliding-window PF implementations (RNAplfold)

- Add SWIG interfaces for sliding-window MFE/PF implementations

- Add proper SWIG interface for alignment and structure plotting functions

- Add proper SWIG interface for duplexfold, duplex_subopt, and its comparative variants

- Add SWIG wrapper for vrna_exp_params_rescale()

- Add explicit destructor for SWIG generated vrna_md_t objects

- Add SWIG perl5 typemap for simple nested STL vectors

- Add dummy field in vrna_structured_domains_s

- Add note about SSE optimized code in reference manual

- Add SWIG interface for findpath implementation

- Add prepare() functions for ptypes-arrays and vrna_(exp_)param_t

- Add warnings for ignored commands in function vrna_commands_apply()

- Add callback featured functions for sliding window MFE and PF implementations

- Change default behavior of adding soft constraints to a vrna_fold_compound_t (store only)

- Several fixes with respect to G-Quadruplex prediction in sliding-window MFE recursions (single sequence and comparative implementation)

- Replace comparative sliding-window MFE recursions (All hits are reported to callback and can be filtered in a post-processing step)

- API: Remove E_mb_loop_stack() and introduce new function vrna_E_mb_loop_stack() as a replacement

- API: change data type of all constraint bit-flags from 'char' to 'unsigned char'

- API: change data type of a2s array in comparative structure prediction from 'unsigned short' to 'unsigned int'

- API: Change function parameter order in vrna_probs_window() to follow the style of other callback-aware functions in RNAlib

- Move sliding-window MFE implementations to new file mfe_window.c

- Fix building PDF Reference manual with non-standard executable paths

- Fix redefinition of macro ON_SAME_STRAND() in subopt.c

- Fix dangling end issues in sliding-window MFE implementations

- Fix regression for --canonicalBPonly switch in RNAfold/RNAcofold/RNAsubopt

- Fix building sliding-window MFE implementation without SVM support

- Fix parsing of STOCKHOLM 1.0 MSA files that contain MSA spanning multiple blocks

- Fix Alidot link in RNAalifold manpage

- Fix wrong pre-processor flags when enabling single-precision PF computations

- Fix unit testing perl5 interface by including builddir/tests in PERL5LIB path

- Fix buffer overflow in hairpin loop sequence motif extraction for circular RNAs

- Fix out-of-bounds memory access in neighbor.c

- Restore capability to compile stand-alone findpath utility

- Restore capability to use non-standard alphabets for structure prediction

- Restore old-API random number functions in SWIG interface

- Allow additional control characters in MAF MSA input that do not end a block

- Improve reference manual

- Make functions in [pair_mat.h](pair_mat.h) static inline

- Prevent users from adding out-of-range base pair soft constraints

- Inline print functions in color_output.inc

- Start documenting callback features in reference manual

- Re-write large portions of sliding-window PF implementation

- Introduce soft-constraint state flag

- Clean-up SWIG unit test framework

- Remove obsolete scripts ct2b.pl and colorrna.pl from src/Utils directory

- Remove old RNAfold tutorial

v2.3.5

- Fix duplication of output filename prefix in RNAfold

- Add V3.0 API for sliding window partition function (a.k.a. RNAPLfold)

- Add G-Quadruplex prediction to RNALalifold

- Add SWIG wrappers for callback-based sliding window comparative MFE prediction

- Add SSE4.1 multiloop decomposition for single sequence MFE prediction

- Enable RNAfold unit tests to run in paralllel

- Enable users to turn-off base pair probability computations in RNAcofold with -a option

- Split move set in neighbor.c

v2.3.4

- Fix G-Quadruplex probability computation for single sequences

- Fix double-free when using SHAPE reactivity data in RNAalifold

- Fix out-of-bounds access in strand_number array

- Fix weighting of SHAPE reactivity data in consensus structure prediction when fewer data than sequences are present

- Fix z-score output in RNALfold

- Substitute field name 'A0'/'B0' in data structure [vrna_dimer_conc_s](vrna_dimer_conc_s) by 'Ac_start'/'Bc_start' to avoid clashes with termios.h (Mac OSX Python wrapper bug)

- Minimize usage of 'unsafe' sprintf() calls

- Enhance auto-id feature in executable programs

- Always sanitize output file names to avoid problems due to strange FASTA headers

- Lift restrictions of FASTA header length in RNAfold, RNAcofold, and RNAeval

- Add ViennaRNA/config.h with pre-processor definitions of configure time choices

- Add test-suite for RNAfold

- Add functions to procude colored EPS structure alignments

- Add function to write Stockholm 1.0 formatted alignments

- Add function to sanitize file names

- Add callback based implementation for sliding-window MFE prediction (single sequences, comparative structure prediction)

- Add fast API 3.0 implementations to generate structural neighbors and perform steepest descent / random walks (Thanks to Gregor!)

- Add parameter option to RNALalifold for colored EPS structure alignment and structure plot output

- Add parameter option to RNALalifold to write hits into Stockholm file

- Add parameter option to RNAalifold to write Stockholm 1.0 formatted output

- Add parameter option to RNAalifold to suppress stderr spam

- Add auto-id feature to RNAplot, RNALfold, RNAsubopt, RNAplfold, RNAheat

- Add SHAPE reactivity derived pseudo-energies as separate output in RNAalifold

- Add colored output to RNA2Dfold, RNALalifold, RNALfold, RNAduplex, RNAheat, RNAinverse, RNAplfold, and RNAsubopt

- Add command line parameters to RNAsubopt to allow for specification of input/output files

v2.3.3

- Fix multiloop contributions for comparative partition function

- Fix building python2 extension module for OSX

v2.3.2

- Fix pair probability plist creation with G-Quadruplexes

- Allow for specification of python2/3-config at configure time

- Fix init of vrna_md_t data structure after call to set_model_details()

- Fix bug in consensus partition function with hard constraints that force nucleotides to be paired

- Fix compilation of functions that use ellipsis/va_list

- Enable generic hard constraints by default

- Fix init of partition function DP matrices for unusually short RNAs

- Fix behavior of RNAplfold for unusually short RNAs

- Report SCI of 0 in RNAalifold when sum of single sequence MFEs is 0

- Avoid multiple includes of pair_mat.h

- Add configure flag to build entirely static executables

v2.3.1

- Add description for how to use unstructured domains through command files to reference manual and RN←
Afold manpage

- Fix compilation issue for Windows platforms with MingW

- Add missing newline in non-TTY-color output of vrna_message_info()

- Fix regression in vrna_md_update() that resulted in incomplete init of reverse-basepair type array

- Extend coverage of generic hard constraints for partition function computations

- Fix scaling of secondary structure in EPS plot such that it always fits into bounding box

- Several fixes and improvements for SWIG generated scripting language interface(s)

v2.3.0

- Add grammar extension with structured and unstructured domains

- Add default implementation for unstructured domains to allow for ligand/protein binding to unpaired structure segments (MFE and PF for single sequences)

- Introduced command files that subsume constraint definition files (currently used in RNAfold and RNAcofold)

- Replace explicit calls to asprintf() with portable equivalent functions in the library

- Fix configure script to deal with situations where Perl module can't be build

- Fix bug in doc/Makefile.am that prevented HTML installation due to long argument list

- Added utility functions that deal with conversion between different units

- Bugfix in SWIG wrapped generic soft constraint feature

- Add subopt() and subopt_zuker() methods to SWIG wrapped fold_compound objects

- Bugfix multiloop decomposition in MFE for circular RNAs

- Add separate function to compute pscore for alignments

- Renamed VRNA_VC_TYPE_∗ macros to VRNA_FC_TYPE_∗

- Bugfix regression that prevented programs to fail on too long input sequences

- Extend EPS dot-plot in RNAfold to include motif/binding probabilities from unstructured domains

- Add variadic functions for error/warning/info message

- Add ID manipulation feature to RNAeval

- Extend API for soft constraint feature for more fine-grained control

- Add section on SWIG wrapped functions in reference manual

- Fix bug in interior loop computations when hard constraints result in non-canonical base pairs

v 2.2.10

- Do not 'forget' subopt results when output is not written to file handle and sorting is switched off

- Fix bad memory access in vrna_subopt() with sorted output

- Add SWIG wrappers for vrna_subopt_cb()

- Correctly show if C11 features are activated in configure status

- Fix autoconf checks to allow for cross compilation again

v 2.2.9

- Fix bug in partition function scaling for backward compatibility of ali_pf_fold()

- Stabilize v3.0 API when building RNAlib and third party program linking against it with compilers that use different C/C++ standards

- Add details on how to link against RNAlib to the reference manual

- Fix RNAlib2.pc

- Fix bug for temperature setting in RNAplfold

- Use -fflat-lto-objects for static RNAlib library to allow linking without LTO

- Fix interpretation of 'P' hard constraint for single nucleotides in constraint definition files

- Add 'A' command for hard constraints

- Fix several hard constraint corner-cases in MFE and partition function computation when nucleotides must not be unpaired

- Fix order of hard constraints when read from input file

- Allow for non-canonical base pairs in MFE and partition function computations if hard constraints demand it

- Fix behavior of --without-swig configure script option

- Fix bug in hard constraints usage of exterior loop MFE prediction with odd dangles

- Add parsers for Clustal, Stockholm, FASTA, and MAF formatted alignment files

- Enable RNAalifold to use Clustal, Stockholm, FASTA, or MAF alignments as input

- Lift restriction of sequence number in alignments for RNAalifold

- Enable ANSI colors for TTY output in RNAfold, RNAcofold, RNAalifold, RNAsubopt, and warnings/errors issued by RNAlib

- Add various new commandline options to manipulate sequence/alignment IDs in RNAfold, RNAcofold and RNAalifold

v 2.2.8

- Fix bad memory access in RNAalifold

- Fix regression in RNAalifold to restore covariance contribution ratio determination for circular RNA alignments

- Changed output of RNAsubopt in energy-band enumeration mode to print MFE and energy range in kcal/mol instead of 10cal/mol

- Include latest Kinfold sources that make use of v3.0 API, therefore speeding up runtime substantially

- Re-activate warnings in RNAeval when non-canonical base pairs are encountered

- Fix syntactic incompatibilities that potentially prevented compilation with compilers other than gcc

- dd function to compare nucleotides encoded in IUPAC format

- Fix regression in energy evaluation for circular RNA sequences

- Fix regression in suboptimal structure enumeration for circular RNAs

- Allow for P i-j k-l commands in constraint definition files

- Make free energy evaluation functions polymorphic

- Add free energy evaluation functions that allow for specifying verbosity level

- Secure functions in alphabet.c against NULL pointer arguments

- Fix incompatibility with swig $>=$ 3.0.9

- Fix memory leak in swig-generated scripting language interface(s) for user-provided target language soft-constraint callbacks

- Expose additional functions to swig-generated scripting language interface(s)

- Build Python3 interface by default

- Start of more comprehensive scripting language interface documentation

- Fix linking of python2/python3 interfaces when libpython is in non-standard directory

- Restructured viennarna.spec for RPM based distributions

- Several syntactic changes in the implementation to minimize compiler warnings

- Fix --with-∗/--without-∗ and --enable-∗/--disable-∗ configure script behavior

v 2.2.7

- Fix partition function scaling for long sequences in RNAfold, RNAalifold, and RNAup

- Fix backtracking issue in RNAcofold when --noLP option is activated

- Fix hard constraints issue for circular RNAs in generating suboptimal structures

- Rebuild reference manual only when actually required

v 2.2.6

- Plugged memory leak in RNAcofold

- Fixed partition function rescaling bug in RNAup

- Fixed bug in RNALfold with window sizes larger than sequence length

- Re-added SCI parameter for RNAalifold

- Fixed backtracking issue for large G-quadruplexes in RNAalifold

- Fixed missing FASTA id in RNAeval output

- Added option to RNAalifold that allows to specify prefix for output files

- Several fixes and additional functions/methods in scripting language interface(s)

- Added version information for scripting language interface(s)

- Some changes to allow for compilation with newer compilers, such as gcc 6.1

v 2.2.5

- Fixed regression in RNAcofold that prohibited output of concentration computations

- Fixed behavior of RNAfold and RNAcofold when hard constraints create empty solution set (programs now abort with error message)

- Added optional Python 3 interface

- Added RNA::Params Perl 5 sub-package

- Update RNA::Design Perl 5 sub-package

- Simplified usage of v3.0 API with default options

- Wrap more functions of v3.0 API in SWIG generated scripting language interfaces

- Plugged some memory leaks in SWIG generated scripting language interfaces

- Changed parameters of recursion status callback in vrna_fold_compound_t

- Enable definition and binding of callback functions from within SWIG target language

- Added optional subpackage Kinwalker

- Added several configure options to ease building and packaging under MacOS X

- Added new utility script RNAdesign.pl

v 2.2.4

- Fixed bug in RNAsubopt that occasionally produced cofolded structures twice

- Removed debugging output in preparations of consensus structure prediction datastructures

v 2.2.3

- Added postscipt annotations for found ligand motifs in RNAfold

- Added more documentation for the constraints features in RNAfold and RNAalifold

- Restore backward compatibility of get_alipf_arrays()

v 2.2.2

- Fix regression bug that occasionally prevented backtracking with RNAcofold --noLP

v 2.2.1

- Fix regression bug that made RNAcofold -a unusable

- Fix regression bug that prohibited RNAfold to compute the MEA structure when G-Quadruplex support was switched on

- Fix bug in Kinfold to enable loading energy parameters from file

- Fix potential use of uninitialized value in RNApdist

- Add manpage for ct2db

- Fix MEA computation when G-Quadruplex support is activated

- Allow for vendor installation of the perl interface using INSTALLDIRS=vendor at configure time

- Install architecture dependent and independent files of the perl and python interface to their correct file system locations

v 2.2.0

- RNAforester is now of version 2.0

- New program RNApvmin to compute pseudo-energy pertubation vector that minimizes discrepancy between observed and predicted pairing probabilities

- SHAPE reactivity support for RNAfold, RNAsubopt, and RNAalifold

- Ligand binding to hairpin- and interior-loop motif support in RNAfold

- New commandline option to limit maximum base pair span for RNAfold, RNAsubopt, RNAcofold, and RN↩ Aalifold

- Bugfix in RNAheat to remove numerical instabilities

- Bugfix in RNAplex to allow for computation of interactions without length limitation

- Bugfix in RNAplot for simple layouts and hairpins of size 0

- (generic) hard- and soft-constraints for MFE, partition function, base pair probabilities, stochastic backtracking, and suboptimal secondary structures of single sequences, sequence alignments, and sequence dimers

- libsvm version as required for z-scoring in RNALfold is now 3.20

- Stochastic backtracking for single sequences is faster due to usage of Boustrophedon scheme

- First polymorphic functions vrna_mfe(), vrna_pf(), and vrna_pbacktrack().

- The FLT_OR_DBL macro is now a typedef

- New functions to convert between different secondary structure representations, such as helix lists, and R↩ NAshapes abstractions

- First object-oriented interface for new API functions in the scripting language interfaces

- new ViennaRNA-perl submodule that augments the Perl interface to RNAlib

- Ligand binding to hairpin- and interior-loop motif support in C-library and scripting language interfaces.

- Libraries are generated using libtool

- Linking of libraries and executables defaults to use Link Time Optimization (LTO)

- Large changes in directory structure of the source code files

v 2.1.9

- Fixed integer underflow bug in RNALfold

- Added Sequence Conservation index (SCI) option to RNAalifold

- Fixed bug in energy evaluation of dangling ends / terminal mismatches of exterior loops and multibranch loops

- Fixed bug in alifold partition function for circular RNAs

- Fixed bug in alifold that scrambled backtracing with activated G-Quadruplex support

- Fixed bug in alifold backtracking for larger G-Quadruplexes

v 2.1.8

- Repaired incorporation of RNAinverse user provided alphabet

- Fix missing FASTA ID in RNAeval output

- prevent race condition in parallel calls of Lfold()

- Fixed memory bug in Lfold() that occured using long sequences and activated G-Quad support

- Added latest version of switch.pl

v 2.1.7

- Fixed bug in RNALfold -z

- Python and Perl interface are compiling again under MacOSX

- Fixed handling of C arrays in Python interface

- Added latest version of switch.pl

- Make relplot.pl work with RNAcofold output

v 2.1.6

- New commandline switches allow for elimination of non-canonical base pairs from constraint structures in RNAfold, RNAalifold and RNAsubopt

- updated moveset functions

- final fix for discrepancy of tri-loop evaluation between partition function and mfe

- pkg-config file now includes the OpenMP linker flag if necessary

- New program ct2db allows for conversion of .ct files into dot-bracket notation (incl. pseudo-knot removal)

v 2.1.5

- Fix for discrepancy between special hairpin loop evaluation in partition functions and MFE

v 2.1.4

- Fix of G-quadruplex support in subopt()

- Fix for discrepancy between special hairpin loop evaluation in partition functions and MFE

v 2.1.3

- RNAfold: Bugfix for ignoring user specified energy parameter files

- RNAcofold: Bugfix for crashing upon constrained folding without specifying a constraint structure

- RNAsubopt: Added G-quadruplex support

- RNAalifold: Added parameter option to specify base pair probability threshold in dotplot

- Fix of several G-quadruplex related bugs

- Added G-quadruplex support in subopt()

v 2.1.2

- RNAfold: Bugfix for randomly missing probabilities in dot-plot during batch job execution

- RNAeval: Bugfix for misinterpreted G-quadruplex containing sequences where the quadruplex starts at nucleotide 1

- RNAsubopt: Slight changes to the output of stochastic backtracking and zuker subopt

- Fix of some memory leaks

- Bugfixes in zukersubopt(), assign_plist_from_pr()

- New threadsafe variants of putoutpU_prob∗() for LPfold()

- Provision of python2 interface support.

v 2.1.1

- Bugfix to restore backward compatibility with ViennaRNA Package 1.8.x API (this bug also affected proper usage of the the perl interface)

v 2.1.0

- G-Quadruplex support in RNAfold, RNAcofold, RNALfold, RNAalifold, RNAeval and RNAplot

- LPfold got a new option to output its computations in split-mode

- several G-Quadruplex related functions were introduced with this release

- several functions for moves in an RNA landscape were introduced

- new function in alipfold.c now enables access to the partition function matrices of alipf_fold()

- different numeric approach was implement for concentration dependend co-folding to avoid instabilities which occured under certain circumstances

v 2.0.7

- Bugfix for RNAplfold where segfault happened upon usage of -O option

- Corrected misbehavior of RNAeval and RNAplot in tty mode

v 2.0.6

- Bugfix for bad type casting with gcc under MacOSX (resulted in accidental "sequence too long" errors)

- Bugfix for disappearing tri-/hexaloop contributions when read in from certain parameter files

- Bugfix for RNALfold that segfaulted on short strange sequences like AT+ repeats

- Change of RNA2Dfold output format for stochastic backtracking

v 2.0.5

- Restored z-score computation capabilities in RNALfold

v 2.0.4

- Bugfix for RNAcofold partition function

- Perl wrapper compatibility to changed RNAlib has been restored

- Backward compatibility for partition function calls has been restored

v 2.0.3

- Bugfix for RNAalifold partition function and base pair probabilities in v2.0.3b

- Added Boltzmann factor scaling in RNAsubopt, RNAalifold, RNAplfold and RNAcofold

- Bugfix for alipfold() in v2.0.3b

- Restored threadsafety of folding matrix access in LPfold.c, alipfold.c, part_func.c, part_func_co.c and part↩
  _func_up.c

- Added several new functions regarding threadsafe function calls in terms of concurrently changing the model
  details

- Added pkg-config file in the distribution to allow easy checks for certain RNAlib2 versions, compiler flags and
  linker flags.

v 2.0.2

- added support for Boltzmann factor scaling in RNAfold

- fixed fastaheader to filename bug

- plugged some memory leaks

v 2.0.1

- First official release of version 2.0

- included latest bugfixes

__History__

2011-03-10 Ronny Lorenz `ronny@tbi.univie.ac.at`

- new naming scheme for all shipped energy parameter files

- fixed bugs that appear while compiling with gcc under MacOS X

- fixed bug in RNAup --interaction-first where the longer of the first two sequences was taken as target

- added full FASTA input support to RNAfold, RNAcofold, RNAheat, RNAplfold RNALfoldz, RNAsubopt and
  RNALfold

2010-11-24 Ronny Lorenz `ronny@tbi.univie.ac.at`

- first full pre-release of version 2.0

2009-11-03 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Fix memory corruption in PS_color_aln()

2009-09-09 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Fix bug in RNAplfold when -u and -L parameters are equal

- Fix double call to [free_arrays()](#) in RNAfold.c

- Improve drawing of cofolded structures

2009-05-14 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Fix occasional segfault in RNAalifold's print_aliout()

2009-02-24 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Add -MEA options to RNAfold and RNAalifold

- change energy_of_alistruct to return float not void

2009-02-24 Ivo Hofacker `ivo@tbi.univie.ac.at`

- RNAfold will draw structures unless -noPS is used (no more "structure too long" messages)

- Restore the "alifold.out" output from RNAalifold -p

- RNAalifold -circ did not work due to wrong return type

- Accessibility calculation with RNAplfold would give wrong results for u$<$=30

2008-12-03 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Add zuker style suboptimals to RNAsubopt (-z)

- [get_line()](#) should be much faster when reading huge sequences (e.g. whole chromosomes for RNALfold)

2008-08-12 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Add Ribosum matrices for covariance scoring in RNAalifold

2008-06-27 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Change RNAalifold to used berni's new energy evaluation w/o gaps

- Add stochastic backtracking in RNAalifold

2008-07-04 Ivo Hofacker `ivo@tbi.univie.ac.at`

- modify output of RNAup (again). Program reading RNAup output will have to updated!

2008-07-02 Ivo Hofacker `ivo@tbi.univie.ac.at`

- RNAplfold now computes accessibilities for all regions up to a max length simultaneously. Slightly slower when only 1 value is needed, but much faster if all of them are wanted. This entails a new output format. Programs reading accessibility output from RNAplfold need to be updated!

2008-03-31 Stephan Bernhart `berni@tbi.univie.ac.at`

- add cofolding to RNAsubopt

2008-01-08 Ivo Hofacker `ivo@tbi.univie.ac.at`

- ensure circfold works even for open chain

2007-12-13 Ulli Mueckstein `ulli@tbi.univie.ac.at`

- upate RNAup related files RNAup can now include the intramolecular structure of both molecules and handles constraints.

2007-12-05 Ronny Lorenz `ronny@tbi.univie.ac.at`

- add circfold variants in part_func.c alipfold.c subopt.c

2007-09-19 Ivo Hofacker `ivo@tbi.univie.ac.at`

- compute the controid structure of the ensemble in RNAfold -p
- fix a missing factor 2 in mean_bp_dist(). CAUTION ensemble diversities returned by RNAfold -p are now twice as large as in earlier versions.

2007-09-04 Ivo Hofacker `ivo@blini.tbi.univie.ac.at`

- fix a bug in Lfold() where base number n-max-4 would never pair

2007-08-26 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add RNAaliduplex the alignment version of RNAduplex
- introduce a minimal distance between hits produced by duplex_subopt()

2007-07-03 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add a loop_energy() function to compute energy of a single loop

2007-06-23 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add aliLfold() and RNALalifold, alignment variant of Lfold()

2007-04-30 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add RNAup to distribution

2007-04-15 Ivo Hofacker `ivo@tbi.univie.ac.at`

- fix segfault in colorps output (thanks to Andres Varon)

2007-03-03 Ivo Hofacker `ivo@tbi.univie.ac.at`

- avoid unnormalized doubles in scale[], big speedup for pf_fold() on very long sequences

2007-02-03 Ivo Hofacker `ivo@tbi.univie.ac.at`

- RNAalifold can now produce colored structure plots and alignment plots

2007-02-01 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Fix segfault in RNAplfold because of missing prototype

2006-12-01 Ivo Hofacker `ivo@tbi.univie.ac.at`

- RNAduplex would segfault when no structure base pairs are possible

2006-08-22 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add computation stacking probabilities using RNAfold -p2
- add -noPS option for NRAfold to supress drawing structures

2006-08-09 Stephan Bernhart `berni@tbi.univie.ac.at`

- RNAplfold can now compute probabilites of unpaired regions (scanning version of RNAup)

2006-06-14 Ivo Hofacker `ivo@tbi.univie.ac.at`

- compile library with -fpic (if available) for use as shared library in the Perl module.
- fix another bug when calling Lfold() repeatedly
- fix switch cmdline parsing in RNAalifold (-mis implied -4)
- fix bug in cofold() with dangles=0

2006-05-08 Ivo Hofacker `ivo@tbi.univie.ac.at`

- fix segfault in Lfold() when calling repeatedly

- fix structure parsing in RNAstruct.c (thanks to Michael Pheasant for reporting both bugs)

- add duplexfold() and alifold() to Perl module

- distinguish window size and max pair span in LPfold

2006-04-05 Ivo Hofacker `ivo@tbi.univie.ac.at`

- fix performance bug in co_pf_fold()

- use relative error for termination of Newton iteration

2006-03-02 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add circular folding in alifold()

2006-01-18 Ivo Hofacker `ivo@tbi.univie.ac.at`

- cleanup berni partition cofold code, including several bug fixes

2006-01-16 Ivo Hofacker `ivo@tbi.univie.ac.at`

- update RNAplfold to working version

- add PS_dot_plot_turn() in PS_dot.c

2005-11-07 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add new utilities colorna and coloraln

2005-10-11 Christoph Flamm `xtof@tbi.univie.ac.at`

- adapt PS_rna_plot() for drawing co-folded structures

2005-07-24 Ivo Hofacker `ivo@tbi.univie.ac.at`

- fix a few memory problems in structure comparison routines

2005-04-30 Ivo Hofacker `ivo@blini.tbi.univie.ac.at`

- add folding of circular RNAs

2005-03-11 Ivo Hofacker `ivo@blini.tbi.univie.ac.at`

- add -mis option to RNAalifold to give "most informative sequence" as consensus

2005-02-10 Ivo Hofacker `ivo@tbi.univie.ac.at`

- move [alifold()](#) into the library

2004-12-22 Stephan Bernhart `berni@tbi.univie.ac.at`

- add partition function version of RNAcofold

2004-12-23 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add RNApaln for fast structural alignments (RNApdist improvement)

2004-08-12 Ivo Hofacker `ivo@tbi.univie.ac.at`

- fix constrained folding in stochastic backtracking

2004-07-21 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add RNAduplex, to compute hybrid structures without intra-molecular pairs

2004-02-09 Ivo Hofacker `ivo@tbi.univie.ac.at`

- fix bug in fold that caused segfaults when using Intel compiler
- add computation of ensemble diversity to RNAfold

2003-09-10 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add annotation options to RNAplot

2003-08-04 Ivo Hofacker `ivo@tbi.univie.ac.at`

- stochastic backtracking finally works. Try e.g. RNAsubopt -p 10

2003-07-18 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add relplot.pl and rotate_ss.pl utilities for reliability annotation and rotation of rna structure plots

2003-01-29 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add RNALfold program to compute locally optimal structures with maximum pair span.
- add RNAcofold for computing hybrid structure

2002-11-07 Ivo Hofacker `ivo@tbi.univie.ac.at`

- change Make_bp_profile() and profile_edit_distance() to use simple (float ∗) arrays; makes Perl access much easier. RNApdist -B now works again

2002-10-28 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Improved Perl module with pod documentation; allow to write things like ($structure, $energy) = RNA↩::fold($seq); Compatibility warning: the ptrvalue() and related functions are gone, see the pod documentation for alternatives.

2002-10-29 Ivo Hofacker `ivo@tbi.univie.ac.at`

- added svg structure plots in PS_dot.c and RNAplot

2002-08-15 Ivo Hofacker `ivo@tbi.univie.ac.at`

- Improve reading of clustal files (alifold)
- add a sample alifold.cgi script

2001-09-18 Ivo Hofacker `ivo@tbi.univie.ac.at`

- moved suboptimal folding into the library, thus it's now accessible from the Perl module

2001-08-31 Ivo Hofacker `ivo@tbi.univie.ac.at`

- added co-folding support in energy_of_struct(), and thus RNAeval

2001-04-30 Ivo Hofacker `ivo@tbi.univie.ac.at`

- switch from handcrafted makefiles to automake and autoconf

2001-04-05 Ivo Hofacker `ivo@tbi.univie.ac.at`

- added PS_rna_plot_a to produce structure plots with annotation

2001-03-03 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add alifold; predict consensus structures from alignment

2000-09-28 Ivo Hofacker `ivo@tbi.univie.ac.at`

- add -d3 option to RNAfold for co-axial stacking

# Chapter 10

# Deprecated List

**Global alifold (const char ∗∗strings, char ∗structure)**

Usage of this function is discouraged! Use vrna_alifold(), or vrna_mfe() instead!

**Global alimake_pair_table (const char ∗structure)**

Use vrna_pt_ali_get() instead!

**Global alipbacktrack (double ∗prob)**

Use vrna_pbacktrack() instead!

**Global alipf_circ_fold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)**

Use vrna_pf() instead

**Global alipf_fold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)**

Use vrna_pf() instead

**Global alipf_fold_par (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl, vrna_exp_param_↩ t ∗parameters, int calculate_bppm, int is_constrained, int is_circular)**

Use vrna_pf() instead

**Global assign_plist_from_db (vrna_ep_t ∗∗pl, const char ∗struc, float pr)**

Use vrna_plist() instead

**Global assign_plist_from_pr (vrna_ep_t ∗∗pl, FLT_OR_DBL ∗probs, int length, double cutoff)**

Use vrna_plist_from_probs() instead!

**Global base_pair**

Do not use this variable anymore!

**Global bondT**

Use vrna_bp_stack_t instead!

**Global bp_distance (const char ∗str1, const char ∗str2)**

Use vrna_bp_distance instead

**Global bppm_symbol (const float ∗x)**

Use vrna_bpp_symbol() instead!

**Global bppm_to_structure (char ∗structure, FLT_OR_DBL ∗pr, unsigned int length)**

Use vrna_db_from_probs() instead!

**Global centroid (int length, double ∗dist)**

This function is deprecated and should not be used anymore as it is not threadsafe!

**Global circalifold (const char ∗∗strings, char ∗structure)**

Usage of this function is discouraged! Use vrna_alicircfold(), and vrna_mfe() instead!

**Global circfold (const char ∗sequence, char ∗structure)**

　　Use vrna_circfold(), or vrna_mfe() instead!

**Global co_pf_fold (char ∗sequence, char ∗structure)**

　　{Use vrna_pf_dimer() instead!}

**Global co_pf_fold_par (char ∗sequence, char ∗structure, vrna_exp_param_t ∗parameters, int calculate_↩**
**bppm, int is_constrained)**

　　Use vrna_pf_dimer() instead!

**Global cofold (const char ∗sequence, char ∗structure)**

　　use vrna_mfe_dimer() instead

**Global cofold_par (const char ∗string, char ∗structure, vrna_param_t ∗parameters, int is_constrained)**

　　use vrna_mfe_dimer() instead

**Global compute_BPdifferences (short ∗pt1, short ∗pt2, unsigned int turn)**

　　Use vrna_refBPdist_matrix() instead

**Global compute_probabilities (double FAB, double FEA, double FEB, vrna_ep_t ∗prAB, vrna_ep_t ∗prA,**
**vrna_ep_t ∗prB, int Alength)**

　　{ Use vrna_pf_dimer_probs() instead!}

**Global constrain_ptypes (const char ∗constraint, unsigned int length, char ∗ptype, int ∗BP, int min_loop↩**
**_size, unsigned int idx_type)**

　　Do not use this function anymore! Structure constraints are now handled through vrna_hc_t and related func-
tions.

**Global copy_pair_table (const short ∗pt)**

　　Use vrna_ptable_copy() instead

**Global cpair**

　　Use vrna_cpair_t instead!

**Global cv_fact**

　　See vrna_md_t.cv_fact, and vrna_mfe() to avoid using global variables

**Global destroy_TwoDfold_variables (TwoDfold_vars ∗our_variables)**

　　Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩
_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**Global destroy_TwoDpfold_variables (TwoDpfold_vars ∗vars)**

　　Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩
_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

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

　　Usage of this function is discouraged! Use vrna_eval_structure(), and vrna_eval_covar_structure() instead!

**Global energy_of_circ_struct (const char ∗string, const char ∗structure)**

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

**Global energy_of_circ_struct_par (const char ∗string, const char ∗structure, vrna_param_t ∗parameters,**
**int verbosity_level)**

　　Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

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

　　Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

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

　　Use vrna_eval_move() instead!

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

　　Use vrna_eval_move_pt() instead!

**Global energy_of_struct (const char ∗string, const char ∗structure)**

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

**Global energy_of_struct_par (const char ∗string, const char ∗structure, vrna_param_t ∗parameters, int verbosity_level)**

Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

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

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

**Global energy_of_struct_pt_par (const char ∗string, short ∗ptable, short ∗s, short ∗s1, vrna_param_↩ t ∗parameters, int verbosity_level)**

Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

**Global energy_of_structure (const char ∗string, const char ∗structure, int verbosity_level)**

Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

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

Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

**Global exp_E_ExtLoop (int type, int si1, int sj1, vrna_exp_param_t ∗P)**

Use vrna_exp_E_ext_stem() instead!

**Global expHairpinEnergy (int u, int type, short si1, short sj1, const char ∗string)**

Use exp_E_Hairpin() from loop_energies.h instead

**Global expLoopEnergy (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)**

Use exp_E_IntLoop() from loop_energies.h instead

**Global export_ali_bppm (void)**

Usage of this function is discouraged! The new vrna_fold_compound_t allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to vrna_pf(), or any of the old API calls for consensus structure partition function folding.

**Global export_circfold_arrays (int ∗Fc_p, int ∗FcH_p, int ∗FcI_p, int ∗FcM_p, int ∗∗fM2_p, int ∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p)**

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**Global export_circfold_arrays_par (int ∗Fc_p, int ∗FcH_p, int ∗FcI_p, int ∗FcM_p, int ∗∗fM2_p, int ∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p, vrna_param_t ∗∗P_p)**

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**Global export_co_bppm (void)**

This function is deprecated and will be removed soon! The base pair probability array is available through the vrna_fold_compound_t data structure, and its associated vrna_mx_pf_t member.

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

folding matrices now reside within the vrna_fold_compound_t. Thus, this function will only work in conjunction with a prior call to the deprecated functions cofold() or cofold_par()

**Global 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)**

folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to cofold() or cofold_par()

**Global export_fold_arrays (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p)**

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**Global export_fold_arrays_par (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype↩ _p, vrna_param_t ∗∗P_p)**

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**Global filecopy (FILE ∗from, FILE ∗to)**

Use vrna_file_copy() instead!

**Global fold (const char ∗sequence, char ∗structure)**

use vrna_fold(), or vrna_mfe() instead!

**Global fold_par (const char ∗sequence, char ∗structure, vrna_param_t ∗parameters, int is_constrained, int is_circular)**

use vrna_mfe() instead!

**Global free_alifold_arrays (void)**

Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced vrna_fold_compound_t is handled by vrna_fold_compound_free()

**Global free_alipf_arrays (void)**

Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with vrna_) will be not affected!

**Global free_arrays (void)**

See vrna_fold(), vrna_circfold(), or vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**Global free_co_arrays (void)**

This function will only free memory allocated by a prior call of cofold() or cofold_par(). See vrna_mfe_dimer() for how to use the new API

**Global free_co_pf_arrays (void)**

This function will be removed for the new API soon! See vrna_pf_dimer(), vrna_fold_compound(), and vrna_fold_compound_free() for an alternative

**Global free_pf_arrays (void)**

See vrna_fold_compound_t and its related functions for how to free memory occupied by the dynamic programming matrices

**Global get_alipf_arrays (short ∗∗∗S_p, short ∗∗∗S5_p, short ∗∗∗S3_p, unsigned short ∗∗∗a2s_p, char ∗∗∗↩ Ss_p, FLT_OR_DBL ∗∗qb_p, FLT_OR_DBL ∗∗qm_p, FLT_OR_DBL ∗∗q1k_p, FLT_OR_DBL ∗∗qln_p, short ∗∗pscore)**

It is discouraged to use this function! The new vrna_fold_compound_t allows direct access to all necessary consensus structure prediction related variables!

**Global get_boltzmann_factor_copy (vrna_exp_param_t ∗parameters)**

Use vrna_exp_params_copy() instead!

**Global get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)**

Use vrna_exp_params() instead!

**Global get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)**

Use vrna_exp_params_comparative() instead!

**Global get_centroid_struct_gquad_pr (int length, double ∗dist)**

This function is deprecated and should not be used anymore as it is not threadsafe!

**Global get_centroid_struct_pl (int length, double ∗dist, vrna_ep_t ∗pl)**

This function was renamed to vrna_centroid_from_plist()

**Global get_centroid_struct_pr (int length, double ∗dist, FLT_OR_DBL ∗pr)**

This function was renamed to vrna_centroid_from_probs()

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

{ Use vrna_pf_dimer_concentrations() instead!}

**Global get_line (FILE ∗fp)**

Use vrna_read_line() as a substitute!

**Global get_monomere_mfes (float ∗e1, float ∗e2)**

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

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

Use vrna_aln_mpi() as a replacement

**Global get_plist (vrna_ep_t *pl, int length, double cut_off)**

{ This function is deprecated and will be removed soon!} use assign_plist_from_pr() instead!

**Global get_scaled_alipf_parameters (unsigned int n_seq)**

Use vrna_exp_params_comparative() instead!

**Global get_scaled_parameters (double temperature, vrna_md_t md)**

Use vrna_params() instead!

**Global get_scaled_pf_parameters (void)**

Use vrna_exp_params() instead!

**Global get_TwoDfold_variables (const char *seq, const char *structure1, const char *structure2, int circ)**

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound←
_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**Global get_TwoDpfold_variables (const char *seq, const char *structure1, char *structure2, int circ)**

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound←
_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

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

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

**Global hamming (const char *s1, const char *s2)**

Use vrna_hamming_distance() instead!

**Global hamming_bound (const char *s1, const char *s2, int n)**

Use vrna_hamming_distance_bound() instead!

**Global iindx**

Do not use this variable anymore!

**Global init_co_pf_fold (int length)**

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

**Global init_pf_fold (int length)**

This function is obsolete and will be removed soon!

**Global init_rand (void)**

Use vrna_init_rand() instead!

**Global initialize_cofold (int length)**

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

**Global initialize_fold (int length)**

See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**Global int_urn (int from, int to)**

Use vrna_int_urn() instead!

**Global Lfold (const char *string, char *structure, int maxdist)**

Use vrna_mfe_window() instead!

**Global Lfoldz (const char *string, char *structure, int maxdist, int zsc, double min_z)**

Use vrna_mfe_window_zscore() instead!

**Global loop_energy (short *ptable, short *s, short *s1, int i)**

Use vrna_eval_loop_pt() instead!

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

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

**Global Make_bp_profile (int length)**

This function is deprecated and will be removed soon! See Make_bp_profile_bppm() for a replacement

**Global make_pair_table (const char *structure)**

Use vrna_ptable() instead

**Global make_pair_table_snoop (const char *structure)**

Use vrna_pt_snoop_get() instead!

**Global make_referenceBP_array (short *reference_pt, unsigned int turn)**

Use vrna_refBPcnt_matrix() instead

**Global mean_bp_dist (int length)**

This function is not threadsafe and should not be used anymore. Use mean_bp_distance() instead!

**Global mean_bp_distance (int length)**

Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

**Global mean_bp_distance_pr (int length, FLT_OR_DBL *pr)**

Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

**Global nc_fact**

See vrna_md_t.nc_fact, and vrna_mfe() to avoid using global variables

**Global nrerror (const char message[])**

Use vrna_message_error() instead!

**Global pack_structure (const char *struc)**

Use vrna_db_pack() as a replacement

**Global PAIR**

Use vrna_basepair_t instead!

**Global pair_info**

Use vrna_pinfo_t instead!

**Global paramT**

Use vrna_param_t instead!

**Global parenthesis_structure (char *structure, vrna_bp_stack_t *bp, int length)**

use vrna_parenthesis_structure() instead

**Global parenthesis_zuker (char *structure, vrna_bp_stack_t *bp, int length)**

use vrna_parenthesis_zuker instead

**Global path_t**

Use vrna_path_t instead!

**Global pbacktrack_circ (char *sequence)**

Use vrna_pbacktrack() instead.

**Global pf_circ_fold (const char *sequence, char *structure)**

Use vrna_pf() instead!

**Global pf_fold_par (const char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate↩_bppm, int is_constrained, int is_circular)**

Use vrna_pf() instead

**Global pf_paramT**

Use vrna_exp_param_t instead!

**Global plist**

Use vrna_ep_t or vrna_elem_prob_s instead!

**Global pr**

Do not use this variable anymore!

**Global print_tty_constraint (unsigned int option)**

Use vrna_message_constraints() instead!

**Global print_tty_constraint_full (void)**

Use vrna_message_constraint_options_all() instead!

**Global print_tty_input_seq (void)**

Use vrna_message_input_seq_simple() instead!

**Global print_tty_input_seq_str (const char ∗s)**

Use vrna_message_input_seq() instead!

**Global PS_dot_plot (char ∗string, char ∗file)**

This function is deprecated and will be removed soon! Use PS_dot_plot_list() instead!

**Global PS_rna_plot (char ∗string, char ∗structure, char ∗file)**

Use vrna_file_PS_rnaplot() instead!

**Global PS_rna_plot_a (char ∗string, char ∗structure, char ∗file, char ∗pre, char ∗post)**

Use vrna_file_PS_rnaplot_a() instead!

**Global PS_rna_plot_a_gquad (char ∗string, char ∗structure, char ∗ssfile, char ∗pre, char ∗post)**

Use vrna_file_PS_rnaplot_a() instead!

**Global random_string (int l, const char symbols[])**

Use vrna_random_string() instead!

**Global read_record (char ∗∗header, char ∗∗sequence, char ∗∗∗rest, unsigned int options)**

This function is deprecated! Use vrna_file_fasta_read_record() as a replacment.

**Global scale_parameters (void)**

Use vrna_params() instead!

**Global sect**

Use vrna_sect_t instead!

**Global set_model_details (vrna_md_t ∗md)**

This function will vanish as soon as backward compatibility of RNAlib is dropped (expected in version 3). Use vrna_md_set_default() instead!

**Global SOLUTION**

Use vrna_subopt_solution_t instead!

**Global space (unsigned size)**

Use vrna_alloc() instead!

**Global st_back**

set the *uniq_ML* flag in vrna_md_t before passing it to vrna_fold_compound().

**Global stackProb (double cutoff)**

Use vrna_stack_prob() instead!

**Global str_DNA2RNA (char ∗sequence)**

Use vrna_seq_toRNA() instead!

**Global str_uppercase (char ∗sequence)**

Use vrna_seq_toupper() instead!

**Global temperature**

Use vrna_md_defaults_temperature(), and vrna_md_defaults_temperature_get() to change, and read the global default temperature settings

**Global time_stamp (void)**

Use vrna_time_stamp() instead!

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

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound←
_TwoD(), vrna_mfe_TwoD(), vrna_backtrack5_TwoD(), and vrna_fold_compound_free() instead!

**Global TwoDfold_vars**

This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**Global TwoDfoldList (TwoDfold_vars ∗vars, int distance1, int distance2)**

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**Global TwoDpfold_pbacktrack (TwoDpfold_vars ∗vars, int d1, int d2)**

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack_TwoD(), and vrna_fold_compound_free() instead!

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

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack5_TwoD(), and vrna_fold_compound_free() instead!

**Class TwoDpfold_vars**

This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

**Global TwoDpfoldList (TwoDpfold_vars ∗vars, int maxDistance1, int maxDistance2)**

Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_compound↩_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

**Global unpack_structure (const char ∗packed)**

Use vrna_db_unpack() as a replacement

**Global update_alifold_params (void)**

Usage of this function is discouraged! The new API uses vrna_fold_compound_t to lump all folding related necessities together, including the energy parameters. Use vrna_update_fold_params() to update the energy parameters within a vrna_fold_compound_t.

**Global update_co_pf_params (int length)**

Use vrna_exp_params_subst() instead!

**Global update_co_pf_params_par (int length, vrna_exp_param_t ∗parameters)**

Use vrna_exp_params_subst() instead!

**Global update_cofold_params (void)**

See vrna_params_subst() for an alternative using the new API

**Global update_cofold_params_par (vrna_param_t ∗parameters)**

See vrna_params_subst() for an alternative using the new API

**Global update_fold_params (void)**

For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

**Global update_fold_params_par (vrna_param_t ∗parameters)**

For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

**Global update_pf_params (int length)**

Use vrna_exp_params_subst() instead

**Global update_pf_params_par (int length, vrna_exp_param_t ∗parameters)**

Use vrna_exp_params_subst() instead

**Global urn (void)**

Use vrna_urn() instead!

**Global VRNA_CONSTRAINT_FILE**

Use 0 instead!

**Global VRNA_CONSTRAINT_MULTILINE**

see vrna_extract_record_rest_structure()

**Global VRNA_CONSTRAINT_NO_HEADER**

This mode is not supported anymore!

**Global VRNA_CONSTRAINT_SOFT_MFE**

This flag has no meaning anymore, since constraints are now always stored!

**Global VRNA_CONSTRAINT_SOFT_PF**

Use VRNA_OPTION_PF instead!

**Global vrna_exp_param_s::id**

This attribute will be removed in version 3

**Global vrna_extract_record_rest_constraint (char ∗∗cstruc, const char ∗∗lines, unsigned int option)**

Use vrna_extract_record_rest_structure() instead!

**Global vrna_fc_s::pscore_pf_compat**

This attribute will vanish in the future!

**Global vrna_fc_s::ptype_pf_compat**

This attribute will vanish in the future! It's meant for backward compatibility only!

**Global warn_user (const char message[])**

Use vrna_message_warning() instead!

**Global xrealloc (void ∗p, unsigned size)**

Use vrna_realloc() instead!

**Global zukersubopt (const char ∗string)**

use vrna_zukersubopt() instead

**Global zukersubopt_par (const char ∗string, vrna_param_t ∗parameters)**

use vrna_zukersubopt() instead

# Chapter 11

# Bug List

**Module domains_up**

Although the additional production rule(s) for unstructured domains as descibed in domains_unstructured are always treated as 'segments possibly bound to one or more ligands', the current implementation requires that at least one ligand is bound. The default implementation already takes care of the required changes, however, upon using callback functions other than the default ones, one has to take care of this fact. Please also note, that this behavior might change in one of the next releases, such that the decomposition schemes as shown above comply with the actual implementation.

**Global vrna_subopt_zuker (vrna_fold_compound_t ∗vc)**

Due to resizing, any pre-existing constraints will be lost!

# Chapter 12

# Module Index

## 12.1  The RNAlib API

Our library is grouped into several modules, each addressing different aspects of RNA secondary structure related problems. You can find an overview of the different groups below.

# Chapter 13

# Data Structure Index

## 13.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 14

# File Index

## 14.1 File List

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

# Chapter 15

# Module Documentation

## 15.1 Free energy evaluation

Functions and variables related to free energy evaluation of sequence/structure pairs.

### 15.1.1 Detailed Description

Functions and variables related to free energy evaluation of sequence/structure pairs.

Collaboration diagram for Free energy evaluation:

**Modules**

- Process and evaluate individual loops

**Files**

- file eval.h

    *Functions and variables related to energy evaluation of sequence/structure pairs.*

**Variables**

- int cut_point

    *set to first pos of second seq for cofolding*
- int eos_debug

    *verbose info from energy_of_struct*

## Basic Energy Evaluation Interface with Dot-Bracket Structure String

- float vrna_eval_structure (vrna_fold_compound_t ∗vc, const char ∗structure)

    *Calculate the free energy of an already folded RNA.*
- float vrna_eval_covar_structure (vrna_fold_compound_t ∗vc, const char ∗structure)

    *Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.*
- float vrna_eval_structure_verbose (vrna_fold_compound_t ∗vc, const char ∗structure, FILE ∗file)

    *Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*
- float vrna_eval_structure_v (vrna_fold_compound_t ∗vc, const char ∗structure, int verbosity_level, FILE ∗file)

    *Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*

## Basic Energy Evaluation Interface with Structure Pair Table

- int vrna_eval_structure_pt (vrna_fold_compound_t ∗vc, const short ∗pt)

    *Calculate the free energy of an already folded RNA.*
- int vrna_eval_structure_pt_verbose (vrna_fold_compound_t ∗vc, const short ∗pt, FILE ∗file)

    *Calculate the free energy of an already folded RNA.*
- int vrna_eval_structure_pt_v (vrna_fold_compound_t ∗vc, const short ∗pt, int verbosity_level, FILE ∗file)

    *Calculate the free energy of an already folded RNA.*

## Simplified Energy Evaluation with Sequence and Dot-Bracket Strings

- float vrna_eval_structure_simple (const char ∗string, const char ∗structure)

    *Calculate the free energy of an already folded RNA.*
- float vrna_eval_circ_structure (const char ∗string, const char ∗structure)

    *Evaluate the free energy of a sequence/structure pair where the sequence is circular.*
- float vrna_eval_gquad_structure (const char ∗string, const char ∗structure)

    *Evaluate the free energy of a sequence/structure pair where the structure may contain G-Quadruplexes.*
- float vrna_eval_circ_gquad_structure (const char ∗string, const char ∗structure)

    *Evaluate the free energy of a sequence/structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*
- float vrna_eval_structure_simple_verbose (const char ∗string, const char ∗structure, FILE ∗file)

    *Calculate the free energy of an already folded RNA and print contributions per loop.*
- float vrna_eval_structure_simple_v (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

    *Calculate the free energy of an already folded RNA and print contributions per loop.*
- float vrna_eval_circ_structure_v (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate free energy of a sequence/structure pair, assume sequence to be circular and print contributions per loop.*
- float vrna_eval_gquad_structure_v (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate free energy of a sequence/structure pair, allow for G-Quadruplexes in the structure and print contributions per loop.*
- float vrna_eval_circ_gquad_structure_v (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate free energy of a sequence/structure pair, assume sequence to be circular, allow for G-Quadruplexes in the structure, and print contributions per loop.*

**Simplified Energy Evaluation with Sequence Alignments and Consensus Structure Dot-Bracket String**

- float vrna_eval_consensus_structure_simple (const char ∗∗alignment, const char ∗structure)

  *Calculate the free energy of an already folded RNA sequence alignment.*

- float vrna_eval_circ_consensus_structure (const char ∗∗alignment, const char ∗structure)

  *Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequences are circular.*

- float vrna_eval_gquad_consensus_structure (const char ∗∗alignment, const char ∗structure)

  *Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the structure may contain G-Quadruplexes.*

- float vrna_eval_circ_gquad_consensus_structure (const char ∗∗alignment, const char ∗structure)

  *Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*

- float vrna_eval_consensus_structure_simple_verbose (const char ∗∗alignment, const char ∗structure, FILE ∗file)

  *Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*

- float vrna_eval_consensus_structure_simple_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)

  *Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*

- float vrna_eval_circ_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity_←
  level, FILE ∗file)

  *Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences and print contributions per loop.*

- float vrna_eval_gquad_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity←
  _level, FILE ∗file)

  *Evaluate the free energy of a consensus structure for an RNA sequence alignment, allow for annotated G-←
  Quadruplexes in the structure and print contributions per loop.*

- float vrna_eval_circ_gquad_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)

  *Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences, allow for annotated G-Quadruplexes in the structure and print contributions per loop.*

**Simplified Energy Evaluation with Sequence String and Structure Pair Table**

- int vrna_eval_structure_pt_simple (const char ∗string, const short ∗pt)

  *Calculate the free energy of an already folded RNA.*

- int vrna_eval_structure_pt_simple_verbose (const char ∗string, const short ∗pt, FILE ∗file)

  *Calculate the free energy of an already folded RNA.*

- int vrna_eval_structure_pt_simple_v (const char ∗string, const short ∗pt, int verbosity_level, FILE ∗file)

  *Calculate the free energy of an already folded RNA.*

**Simplified Energy Evaluation with Sequence Alignment and Consensus Structure Pair Table**

- int vrna_eval_consensus_structure_pt_simple (const char ∗∗alignment, const short ∗pt)

  *Evaluate the Free Energy of a Consensus Secondary Structure given a Sequence Alignment.*

- int **vrna_eval_consensus_structure_pt_simple_verbose** (const char ∗∗alignment, const short ∗pt, FILE ∗file)

- int **vrna_eval_consensus_structure_pt_simple_v** (const char ∗∗alignment, const short ∗pt, int verbosity←
  _level, FILE ∗file)

### Basic Loop Energy Evaluation with Structure Pair Table

- int vrna_eval_loop_pt (vrna_fold_compound_t ∗vc, int i, const short ∗pt)

  *Calculate energy of a loop.*
- int vrna_eval_loop_pt_v (vrna_fold_compound_t ∗vc, int i, const short ∗pt, int verbosity_level)

  *Calculate energy of a loop.*

### Basic Energy Evaluation for atomic moves

Here, atomic moves are considered opening or closing a single base pair, as well as shifting one pairing partner of an existing pair to a different location

- float vrna_eval_move (vrna_fold_compound_t ∗vc, const char ∗structure, int m1, int m2)

  *Calculate energy of a move (closing or opening of a base pair)*
- int vrna_eval_move_pt (vrna_fold_compound_t ∗vc, short ∗pt, int m1, int m2)

  *Calculate energy of a move (closing or opening of a base pair)*
- int **vrna_eval_move_pt_simple** (const char ∗string, short ∗pt, int m1, int m2)
- int **vrna_eval_move_shift_pt** (vrna_fold_compound_t ∗vc, vrna_move_t ∗m, short ∗structure)

### Deprecated Energy Evaluation functions

Using the functions below is discouraged as they have been marked deprecated and will be removed from the library in the (near) future!

- 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, vrna_param_t ∗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, vrna_param_t ∗parameters, int verbosity_level)

  *Calculate the free energy of an already folded circular RNA.*
- float **energy_of_gquad_structure** (const char ∗string, const char ∗structure, int verbosity_level)
- float **energy_of_gquad_struct_par** (const char ∗string, const char ∗structure, vrna_param_t ∗parameters, int verbosity_level)
- 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, vrna_param_t ∗parameters, int verbosity_level)

  *Calculate the free energy of an already folded RNA.*
- 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.*
- 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)

### 15.1.2 Function Documentation

#### 15.1.2.1 vrna_eval_structure()

```
float vrna_eval_structure (
            vrna_fold_compound_t * vc,
            const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given pair of structure and sequence (alignment). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not need to contain any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

**Note**

> Accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE and VRNA_FC_TYPE_COMPARATIVE

**See also**

> vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(), vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_eval_covar_structure()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|
| structure | Secondary structure in dot-bracket notation |

**Returns**

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

**SWIG Wrapper Notes** This function is attached as method **eval_structure()** to objects of type *fold_compound*

#### 15.1.2.2 vrna_eval_covar_structure()

```
float vrna_eval_covar_structure (
            vrna_fold_compound_t * vc,
            const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.

Consensus structure prediction is driven by covariance scores of base pairs in rows of the provided alignment. This function allows one to retrieve the total amount of this covariance pseudo energy scores. The vrna_fold_compound_t does not need to contain any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound_comparative(alignment, NULL,
     VRNA_OPTION_EVAL_ONLY);
```

**Note**

Accepts vrna_fold_compound_t of type VRNA_FC_TYPE_COMPARATIVE only!

**See also**

vrna_fold_compound_comparative(), vrna_eval_structure()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|
| structure | Secondary (consensus) structure in dot-bracket notation |

**Returns**

The covariance pseudo energy score of the input structure given the input sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is attached as method **eval_covar_structure()** to objects of type *fold_↩ compound*

**15.1.2.3 vrna_eval_structure_verbose()**

```
float vrna_eval_structure_verbose (
          vrna_fold_compound_t * vc,
          const char * structure,
          FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA and print contributions on a per-loop base.

This function is a simplyfied version of vrna_eval_structure_v() that uses the *default* verbosity level. (

**See also**

vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(),

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
| --- | --- |
| structure | Secondary structure in dot-bracket notation |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is attached as method **eval_structure_verbose()** to objects of type *fold_↩ compound*

**15.1.2.4 vrna_eval_structure_v()**

```
float vrna_eval_structure_v (
            vrna_fold_compound_t * vc,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

`#include <ViennaRNA/eval.h>`

Calculate the free energy of an already folded RNA and print contributions on a per-loop base.

This function allows for detailed energy evaluation of a given sequence/structure pair. In contrast to vrna_eval_structure() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluting functions, while values $\geq 1$ allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

**See also**

vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(),

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
| --- | --- |
| structure | Secondary structure in dot-bracket notation |
| verbosity_level | The level of verbosity of this function |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**15.1.2.5   vrna_eval_structure_pt()**

```
int vrna_eval_structure_pt (
            vrna_fold_compound_t * vc,
            const short * pt )
```

#include <ViennaRNA/eval.h>

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from vrna_ptable(). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

**See also**

> vrna_ptable(), vrna_eval_structure(), vrna_eval_structure_pt_verbose()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|----|--------------------------------------------------------------------------|
| pt | Secondary structure as pair_table |

**Returns**

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

**SWIG Wrapper Notes**  This function is attached as method **eval_structure_pt()** to objects of type *fold_compound*

**15.1.2.6   vrna_eval_structure_pt_verbose()**

```
int vrna_eval_structure_pt_verbose (
            vrna_fold_compound_t * vc,
            const short * pt,
            FILE * file )
```

#include <ViennaRNA/eval.h>

Calculate the free energy of an already folded RNA.

This function is a simplyfied version of vrna_eval_structure_simple_v() that uses the *default* verbosity level.

**See also**

vrna_eval_structure_pt_v(), vrna_ptable(), vrna_eval_structure_pt(), vrna_eval_structure_verbose()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|----|--------------------------------------------------------------------------|
| pt | Secondary structure as pair_table |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is attached as method **eval_structure_pt_verbose()** to objects of type *fold↩ _compound*

**15.1.2.7 vrna_eval_structure_pt_v()**

```
int vrna_eval_structure_pt_v (
            vrna_fold_compound_t * vc,
            const short * pt,
            int verbosity_level,
            FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from vrna_ptable(). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

In contrast to vrna_eval_structure_pt() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive verbosity↩ _level activates potential warning message of the energy evaluting functions, while values $\geq 1$ allow for detailed control of what data is printed. A negative parameter verbosity_level turns off printing all together.

**See also**

vrna_ptable(), vrna_eval_structure_pt(), vrna_eval_structure_verbose()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|----|--------------------------------------------------------------------------|
| pt | Secondary structure as pair_table |
| verbosity_level | The level of verbosity of this function |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**15.1.2.8 vrna_eval_structure_simple()**

```
int vrna_eval_structure_simple (
            const char * sequence,
            const char * structure )
```

#include <ViennaRNA/eval.h>

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair. In contrast to vrna_eval_structure() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**See also**

> vrna_eval_structure(), vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose(),

**Parameters**

| string | RNA sequence in uppercase letters |
|---|---|
| structure | Secondary structure in dot-bracket notation |

**Returns**

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

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for vrna_eval_structure_simple_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.9 vrna_eval_circ_structure()**

```
int vrna_eval_circ_structure (
            const char * sequence,
            const char * structure )
```

#include <ViennaRNA/eval.h>

Evaluate the free energy of a sequence/structure pair where the sequence is circular.

**See also**

> vrna_eval_structure_simple(), vrna_eval_gquad_structure(), vrna_eval_circ_consensus_structure(), vrna_eval_circ_structure_v
> vrna_eval_structure()

**Parameters**

| | |
|---|---|
| *string* | RNA sequence in uppercase letters |
| *structure* | Secondary structure in dot-bracket notation |

**Returns**

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

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for vrna_eval_circ_structure_v()
and, thus, allows for two additional, optional arguments, the verbosity level and a file handle
which default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.10 vrna_eval_gquad_structure()**

```
int vrna_eval_gquad_structure (
            const char * sequence,
            const char * structure )
```

#include <ViennaRNA/eval.h>

Evaluate the free energy of a sequence/structure pair where the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be
denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**See also**

> vrna_eval_structure_simple(), vrna_eval_circ_structure(), vrna_eval_gquad_consensus_structure(), vrna_eval_gquad_structure
> vrna_eval_structure()

**Parameters**

| | |
|---|---|
| *string* | RNA sequence in uppercase letters |
| *structure* | Secondary structure in dot-bracket notation |

**Returns**

> The free energy of the structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for vrna_eval_gquad_structure_v()
and, thus, allows for two additional, optional arguments, the verbosity level and a file handle
which default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.11 vrna_eval_circ_gquad_structure()**

```
int vrna_eval_circ_gquad_structure (
            const char * sequence,
            const char * structure )
```

`#include <`[`ViennaRNA/eval.h`](#)`>`

Evaluate the free energy of a sequence/structure pair where the sequence is circular and the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**See also**

vrna_eval_structure_simple(), vrna_eval_circ_gquad_consensus_structure(), vrna_eval_circ_gquad_structure_v(), vrna_eval_structure()

**Parameters**

| string | RNA sequence in uppercase letters |
|---|---|
| structure | Secondary structure in dot-bracket notation |

**Returns**

The free energy of the structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for vrna_eval_circ_gquad_structure_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.12 vrna_eval_structure_simple_verbose()**

```
int vrna_eval_structure_simple_verbose (
            const char * sequence,
            const char * structure,
            FILE * file )
```

`#include <`[`ViennaRNA/eval.h`](#)`>`

Calculate the free energy of an already folded RNA and print contributions per loop.

This function is a simplyfied version of vrna_eval_structure_simple_v() that uses the *default* verbosity level.

**See also**

vrna_eval_structure_simple_v(), vrna_eval_structure_verbose(), vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose()

**Parameters**

| *string* | RNA sequence in uppercase letters |
|---|---|
| *structure* | Secondary structure in dot-bracket notation |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is not available. Use vrna_eval_structure_simple_v() instead!

**15.1.2.13 vrna_eval_structure_simple_v()**

```
int vrna_eval_structure_simple_v (
            const char * sequence,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

`#include <ViennaRNA/eval.h>`

Calculate the free energy of an already folded RNA and print contributions per loop.

This function allows for detailed energy evaluation of a given sequence/structure pair. In contrast to vrna_eval_structure() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluting functions, while values $\geq 1$ allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

In contrast to vrna_eval_structure_verbose() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Threefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**See also**

> vrna_eval_structure_verbose(), vrna_eval_structure_pt(), vrna_eval_structure_pt_verbose(),

**Parameters**

| *string* | RNA sequence in uppercase letters |
|---|---|
| *structure* | Secondary structure in dot-bracket notation |
| *verbosity_level* | The level of verbosity of this function |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

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

This function is available through an overloaded version of vrna_eval_structure_simple().
The last two arguments for this function are optional and default to #VRNA_VERBOSITY↩
_QUIET and NULL, respectively.

**15.1.2.14    vrna_eval_circ_structure_v()**

```
int vrna_eval_circ_structure_v (
            const char * sequence,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

`#include <ViennaRNA/eval.h>`

Evaluate free energy of a sequence/structure pair, assume sequence to be circular and print contributions per loop.

This function is the same as vrna_eval_structure_simple_v() but assumes the input sequence to be circularized.

**See also**

vrna_eval_structure_simple_v(), vrna_eval_circ_structure(), vrna_eval_structure_verbose()

**Parameters**

| | |
|---|---|
| *string* | RNA sequence in uppercase letters |
| *structure* | Secondary structure in dot-bracket notation |
| *verbosity_level* | The level of verbosity of this function |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes**    This function is available through an overloaded version of vrna_eval_circ_structure(). The
last two arguments for this function are optional and default to #VRNA_VERBOSITY_QU↩
IET and NULL, respectively.

**15.1.2.15    vrna_eval_gquad_structure_v()**

```
int vrna_eval_gquad_structure_v (
            const char * sequence,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate free energy of a sequence/structure pair, allow for G-Quadruplexes in the structure and print contributions per loop.

This function is the same as vrna_eval_structure_simple_v() but allows for annotated G-Quadruplexes in the dot-bracket structure input.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**See also**

vrna_eval_structure_simple_v(), vrna_eval_gquad_structure(), vrna_eval_structure_verbose()

**Parameters**

| string | RNA sequence in uppercase letters |
|---|---|
| structure | Secondary structure in dot-bracket notation |
| verbosity_level | The level of verbosity of this function |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is available through an overloaded version of vrna_eval_gquad_structure(). The last two arguments for this function are optional and default to #VRNA_VERBOSITY↩ _QUIET and NULL, respectively.

**15.1.2.16 vrna_eval_circ_gquad_structure_v()**

```
int vrna_eval_circ_gquad_structure_v (
            const char * sequence,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate free energy of a sequence/structure pair, assume sequence to be circular, allow for G-Quadruplexes in the structure, and print contributions per loop.

This function is the same as vrna_eval_structure_simple_v() but assumes the input sequence to be circular and allows for annotated G-Quadruplexes in the dot-bracket structure input.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**Parameters**

| *string* | RNA sequence in uppercase letters |
| --- | --- |
| *structure* | Secondary structure in dot-bracket notation |
| *verbosity_level* | The level of verbosity of this function |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is available through an overloaded version of vrna_eval_circ_gquad_structure(). The last two arguments for this function are optional and default to #VRNA_VERBOSIT⏎ Y_QUIET and NULL, respectively.

**15.1.2.17    vrna_eval_consensus_structure_simple()**

```
int vrna_eval_consensus_structure_simple (
            const char ** alignment,
            const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA sequence alignment.

This function allows for energy evaluation for a given multiple sequence alignment and consensus structure pair. In contrast to vrna_eval_structure() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

vrna_eval_covar_structure(), vrna_eval_structure(), vrna_eval_structure_pt(), vrna_eval_structure_verbose(), vrna_eval_structure_pt_verbose()

**Parameters**

| *alignment* | RNA sequence alignment in uppercase letters and hyphen ('-') to denote gaps |
| --- | --- |
| *structure* | Consensus Secondary structure in dot-bracket notation |

**Returns**

> The free energy of the consensus structure given the input alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloadeded version of vrna_eval_structure_simple(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

### 15.1.2.18 vrna_eval_circ_consensus_structure()

```
int vrna_eval_circ_consensus_structure (
        const char ** alignment,
        const char * structure )
```

#include <ViennaRNA/eval.h>

Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequences are circular.

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_covar_structure(), vrna_eval_consensus_structure_simple(), vrna_eval_gquad_consensus_structure(), vrna_eval_circ_structure(), vrna_eval_circ_consensus_structure_v(), vrna_eval_structure()

**Parameters**

| alignment | RNA sequence alignment in uppercase letters |
|-----------|---------------------------------------------|
| structure | Consensus secondary structure in dot-bracket notation |

**Returns**

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

**SWIG Wrapper Notes** This function is available through an overloadeded version of vrna_eval_circ_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

### 15.1.2.19 vrna_eval_gquad_consensus_structure()

```
int vrna_eval_gquad_consensus_structure (
        const char ** alignment,
        const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_covar_structure(), vrna_eval_consensus_structure_simple(), vrna_eval_circ_consensus_structure(), vrna_eval_gquad_structure(), vrna_eval_gquad_consensus_structure_v(), vrna_eval_structure()

**Parameters**

| alignment | RNA sequence alignment in uppercase letters |
| --- | --- |
| structure | Consensus secondary structure in dot-bracket notation |

**Returns**

> The free energy of the consensus structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloadeded version of vrna_eval_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**15.1.2.20 vrna_eval_circ_gquad_consensus_structure()**

```
int vrna_eval_circ_gquad_consensus_structure (
            const char ** alignment,
            const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequence is circular and the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_covar_structure(), vrna_eval_consensus_structure_simple(), vrna_eval_circ_consensus_structure(), vrna_eval_gquad_structure(), vrna_eval_circ_gquad_consensus_structure_v(), vrna_eval_structure()

**Parameters**

| | |
|---|---|
| *alignment* | RNA sequence alignment in uppercase letters |
| *structure* | Consensus secondary structure in dot-bracket notation |

**Returns**

> The free energy of the consensus structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloadeded version of vrna_eval_circ_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**15.1.2.21 vrna_eval_consensus_structure_simple_verbose()**

```
int vrna_eval_consensus_structure_simple_verbose (
            const char ** alignment,
            const char * structure,
            FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.

This function is a simplyfied version of vrna_eval_consensus_structure_simple_v() that uses the *default* verbosity level.

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_consensus_structure_simple_v(), vrna_eval_structure_verbose(), vrna_eval_structure_pt(), vrna_eval_structure_pt_verbose()

**Parameters**

| alignment | RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-') |
|---|---|
| structure | Consensus secondary structure in dot-bracket notation |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

The free energy of the conensus structure given the aligned input sequences in kcal/mol

**SWIG Wrapper Notes** This function is not available. Use vrna_eval_consensus_structure_simple_v() instead!

**15.1.2.22 vrna_eval_consensus_structure_simple_v()**

```
int vrna_eval_consensus_structure_simple_v (
            const char ** alignment,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

`#include <ViennaRNA/eval.h>`

Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.

This function allows for detailed energy evaluation of a given sequence alignment/consensus structure pair. In contrast to vrna_eval_consensus_structure_simple() this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluting functions, while values $\geq 1$ allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

vrna_eval_consensus_structure(), vrna_eval_structure()

**Parameters**

| alignment | RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-') |
|---|---|
| structure | Consensus secondary structure in dot-bracket notation |
| verbosity_level | The level of verbosity of this function |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is available through an overloaded version of vrna_eval_structure_simple(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.23 vrna_eval_circ_consensus_structure_v()**

```
int vrna_eval_circ_consensus_structure_v (
            const char ** alignment,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

`#include <`ViennaRNA/eval.h`>`

Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences and print contributions per loop.

This function is identical with vrna_eval_consensus_structure_simple_v() but assumed the aligned sequences to be circular.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

vrna_eval_consensus_structure_simple_v(), vrna_eval_circ_consensus_structure(), vrna_eval_structure()

**Parameters**

| | |
|---|---|
| *alignment* | RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-') |
| *structure* | Consensus secondary structure in dot-bracket notation |
| *verbosity_level* | The level of verbosity of this function |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

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

**SWIG Wrapper Notes** This function is available through an overloaded version of vrna_eval_circ_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRN←A_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.24    vrna_eval_gquad_consensus_structure_v()**

```
int vrna_eval_gquad_consensus_structure_v (
            const char ** alignment,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

#include <ViennaRNA/eval.h>

Evaluate the free energy of a consensus structure for an RNA sequence alignment, allow for annotated G-↩ Quadruplexes in the structure and print contributions per loop.

This function is identical with vrna_eval_consensus_structure_simple_v() but allows for annotated G-Quadruplexes in the consensus structure.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_consensus_structure_simple_v(), vrna_eval_gquad_consensus_structure(), vrna_eval_structure()

**Parameters**

| | |
|---|---|
| *alignment* | RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-') |
| *structure* | Consensus secondary structure in dot-bracket notation |
| *verbosity_level* | The level of verbosity of this function |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

> The free energy of the consensus structure given the sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of vrna_eval_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.25 vrna_eval_circ_gquad_consensus_structure_v()**

```
int vrna_eval_circ_gquad_consensus_structure_v (
            const char ** alignment,
            const char * structure,
            int verbosity_level,
            FILE * file )
```

`#include <`ViennaRNA/eval.h`>`

Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences, allow for annotated G-Quadruplexes in the structure and print contributions per loop.

This function is identical with vrna_eval_consensus_structure_simple_v() but assumes the sequences in the alignment to be circular and allows for annotated G-Quadruplexes in the consensus structure.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots ('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++.++
```

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_consensus_structure_simple_v(), vrna_eval_circ_gquad_consensus_structure(), vrna_eval_structure()

**Parameters**

| | |
|---|---|
| *alignment* | RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-') |
| *structure* | Consensus secondary structure in dot-bracket notation |
| *verbosity_level* | The level of verbosity of this function |
| *file* | A file handle where this function should print to (may be NULL). |

**Returns**

> The free energy of the consensus structure given the sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of vrna_eval_circ_gquad_structure(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.26 vrna_eval_structure_pt_simple()**

```
int vrna_eval_structure_pt_simple (
            const char * sequence,
            const short * pt )
```

`#include <`ViennaRNA/eval.h`>`

Calculate the free energy of an already folded RNA.

In contrast to vrna_eval_structure_pt() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Threefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**See also**

> vrna_ptable(), vrna_eval_structure_simple(), vrna_eval_structure_pt()

**Parameters**

| | |
|---|---|
| *string* | RNA sequence in uppercase letters |
| *pt* | Secondary structure as pair_table |

**Returns**

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

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for vrna_eval_structure_pt_v() and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to #VRNA_VERBOSITY_QUIET and NULL, respectively.

**15.1.2.27 vrna_eval_structure_pt_simple_verbose()**

```
int vrna_eval_structure_pt_simple_verbose (
            const char * string,
            const short * pt,
            FILE * file )
```

`#include <`ViennaRNA/eval.h`>`

Calculate the free energy of an already folded RNA.

This function is a simplyfied version of vrna_eval_structure_pt_simple_v() that uses the *default* verbosity level.

**See also**

> vrna_eval_structure_pt_simple_v(), vrna_ptable(), vrna_eval_structure_pt_verbose(), vrna_eval_structure_simple()

**Parameters**

| string | RNA sequence in uppercase letters |
|---|---|
| pt | Secondary structure as pair_table |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**15.1.2.28 vrna_eval_structure_pt_simple_v()**

```
int vrna_eval_structure_pt_simple_v (
            const char * string,
            const short * pt,
            int verbosity_level,
            FILE * file )
```

`#include <ViennaRNA/eval.h>`

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair_table format as obtained from vrna_ptable(). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

In contrast to vrna_eval_structure_pt_verbose() this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Threefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**See also**

vrna_ptable(), vrna_eval_structure_pt_v(), vrna_eval_structure_simple()

**Parameters**

| string | RNA sequence in uppercase letters |
|---|---|
| pt | Secondary structure as pair_table |
| verbosity_level | The level of verbosity of this function |
| file | A file handle where this function should print to (may be NULL). |

**Returns**

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

**15.1.2.29 vrna_eval_consensus_structure_pt_simple()**

```
int vrna_eval_consensus_structure_pt_simple (
            const char ** alignment,
            const short * pt )
```

#include <ViennaRNA/eval.h>

Evaluate the Free Energy of a Consensus Secondary Structure given a Sequence Alignment.

**Note**

> The free energy returned from this function already includes the covariation pseudo energies that is used fir comparative structure prediction within this library.

**See also**

> vrna_eval_consensus_structure_simple(), vrna_eval_structure_pt(), vrna_eval_structure(), vrna_eval_covar_structure()

**Parameters**

| alignment | RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-') |
|-----------|-------------------------------------------------------------------------------|
| pt        | Secondary structure in pair table format                                      |

**Returns**

> Free energy of the consensus structure in 10cal/mol

**SWIG Wrapper Notes** This function is available through an overloadeded version of vrna_eval_structure_pt_simple(). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**15.1.2.30 vrna_eval_loop_pt()**

```
int vrna_eval_loop_pt (
            vrna_fold_compound_t * vc,
            int i,
            const short * pt )
```

#include <ViennaRNA/eval.h>

Calculate energy of a loop.

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|----|---------------------------------------------------------------------------|
| i | position of covering base pair |
| pt | the pair table of the secondary structure |

**Returns**

> free energy of the loop in 10cal/mol

**SWIG Wrapper Notes** This function is attached as method **eval_loop_pt()** to objects of type *fold_compound*

**15.1.2.31 vrna_eval_loop_pt_v()**

```
int vrna_eval_loop_pt_v (
            vrna_fold_compound_t * vc,
            int i,
            const short * pt,
            int verbosity_level )
```

#include <ViennaRNA/eval.h>

Calculate energy of a loop.

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|----|---------------------------------------------------------------------------|
| i | position of covering base pair |
| pt | the pair table of the secondary structure |
| verbosity_level | The level of verbosity of this function |

**Returns**

> free energy of the loop in 10cal/mol

**15.1.2.32 vrna_eval_move()**

```
float vrna_eval_move (
            vrna_fold_compound_t * vc,
            const char * structure,
            int m1,
            int m2 )
```

#include <ViennaRNA/eval.h>

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**

vrna_eval_move_pt()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|
| *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

**SWIG Wrapper Notes**  This function is attached as method **eval_move()** to objects of type *fold_compound*

**15.1.2.33   vrna_eval_move_pt()**

```
int vrna_eval_move_pt (
            vrna_fold_compound_t * vc,
            short * pt,
            int m1,
            int m2 )
```

`#include <`ViennaRNA/eval.h`>`

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**

vrna_eval_move()

**Parameters**

| vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|
| *pt* | the pair table of the secondary structure |
| *m1* | first coordinate of base pair |
| *m2* | second coordinate of base pair |

**Returns**

energy change of the move in 10cal/mol

**SWIG Wrapper Notes**  This function is attached as method **eval_move_pt()** to objects of type *fold_compound*

**15.1.2.34 energy_of_structure()**

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

#include <ViennaRNA/eval.h>

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.

**Deprecated** Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

**See also**

> vrna_eval_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

**15.1.2.35 energy_of_struct_par()**

```
float energy_of_struct_par (
            const char * string,
            const char * structure,
            vrna_param_t * parameters,
            int verbosity_level )
```

#include <ViennaRNA/eval.h>

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

**Deprecated** Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

**See also**

> vrna_eval_structure()

**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

**15.1.2.36 energy_of_circ_structure()**

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

#include <ViennaRNA/eval.h>

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

**Deprecated** Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

**See also**

> vrna_eval_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

**15.1.2.37   energy_of_circ_struct_par()**

```
float energy_of_circ_struct_par (
              const char * string,
              const char * structure,
              vrna_param_t * parameters,
              int verbosity_level )
```

`#include <`ViennaRNA/eval.h`>`

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

**Deprecated**  Use vrna_eval_structure() or vrna_eval_structure_verbose() instead!

**See also**

vrna_eval_structure()

**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

**15.1.2.38   energy_of_structure_pt()**

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

`#include <`ViennaRNA/eval.h`>`

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.

**Deprecated** Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

See also

vrna_eval_structure_pt()

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

**15.1.2.39   energy_of_struct_pt_par()**

```
int energy_of_struct_pt_par (
            const char * string,
            short * ptable,
            short * s,
            short * s1,
            vrna_param_t * parameters,
            int verbosity_level )
```

#include <ViennaRNA/eval.h>

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

**Deprecated** Use vrna_eval_structure_pt() or vrna_eval_structure_pt_verbose() instead!

See also

vrna_eval_structure_pt()

**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

**15.1.2.40 energy_of_move()**

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

`#include <`ViennaRNA/eval.h`>`

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).

**Deprecated** Use vrna_eval_move() instead!

**See also**

vrna_eval_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

### 15.1.2.41 energy_of_move_pt()

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

`#include <`ViennaRNA/eval.h`>`

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).

**Deprecated** Use vrna_eval_move_pt() instead!

**See also**

> vrna_eval_move_pt()

**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

### 15.1.2.42 loop_energy()

```
int loop_energy (
            short * ptable,
            short * s,
            short * s1,
            int i )
```

`#include <`ViennaRNA/eval.h`>`

Calculate energy of a loop.

**Deprecated** Use vrna_eval_loop_pt() instead!

**See also**

> vrna_eval_loop_pt()

**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

**15.1.2.43 energy_of_struct()**

```
float energy_of_struct (
            const char * string,
            const char * structure )
```

`#include <`ViennaRNA/eval.h`>`

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

**15.1.2.44 energy_of_struct_pt()**

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

#include <ViennaRNA/eval.h>

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

**15.1.2.45 energy_of_circ_struct()**

```
float energy_of_circ_struct (
            const char * string,
            const char * structure )
```

#include <ViennaRNA/eval.h>

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

## 15.2 Process and evaluate individual loops

### 15.2.1 Detailed Description

Collaboration diagram for Process and evaluate individual loops:

**Files**

- file exterior_loops.h

  *Energy evaluation of exterior loops for MFE and partition function calculations.*
- file hairpin_loops.h

  *Energy evaluation of hairpin loops for MFE and partition function calculations.*
- file interior_loops.h

  *Energy evaluation of interior loops for MFE and partition function calculations.*
- file loop_energies.h

  *Energy evaluation for MFE and partition function calculations.*
- file multibranch_loops.h

  *Energy evaluation of multibranch loops for MFE and partition function calculations.*

**Data Structures**

- struct vrna_mx_pf_aux_el_t

  *Auxiliary helper arrays for fast exterior loop computations. More...*
- struct vrna_mx_pf_aux_ml_t

  *Auxiliary helper arrays for fast exterior loop computations. More...*

**Functions**

- int vrna_E_ext_stem (unsigned int type, int n5d, int n3d, vrna_param_t ∗p)

  *Evaluate a stem branching off the exterior loop.*
- FLT_OR_DBL vrna_exp_E_ext_stem (unsigned int type, int n5d, int n3d, vrna_exp_param_t ∗p)

  *Evaluate a stem branching off the exterior loop (Boltzmann factor version)*
- int E_Stem (int type, int si1, int sj1, int extLoop, vrna_param_t ∗P)

  *Compute the energy contribution of a stem branching off a loop-region.*
- FLT_OR_DBL exp_E_ExtLoop (int type, int si1, int sj1, vrna_exp_param_t ∗P)
- FLT_OR_DBL exp_E_Stem (int type, int si1, int sj1, int extLoop, vrna_exp_param_t ∗P)
- PRIVATE int E_Hairpin (int size, int type, int si1, int sj1, const char ∗string, vrna_param_t ∗P)

  *Compute the Energy of a hairpin-loop.*
- PRIVATE FLT_OR_DBL exp_E_Hairpin (int u, int type, short si1, short sj1, const char ∗string, vrna_exp_param_t ∗P)

  *Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop.*
- int vrna_E_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.*
- int vrna_E_ext_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.*
- int vrna_eval_ext_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate free energy of an exterior hairpin loop.*
- int vrna_eval_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate free energy of a hairpin loop.*
- FLT_OR_DBL vrna_exp_E_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *High-Level function for hairpin loop energy evaluation (partition function variant)*
- int vrna_BT_hp_loop (vrna_fold_compound_t ∗vc, int i, int j, int en, vrna_bp_stack_t ∗bp_stack, int ∗stack↩
  _count)

  *Backtrack a hairpin loop closed by $(i, j)$.*
- PRIVATE int E_IntLoop (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t ∗P)
- PRIVATE FLT_OR_DBL exp_E_IntLoop (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short
  sq1, vrna_exp_param_t ∗P)
- int vrna_eval_int_loop (vrna_fold_compound_t ∗vc, int i, int j, int k, int l)

  *Evaluate the free energy contribution of an interior loop with delimiting base pairs $(i, j)$ and $(k, l)$.*
- int vrna_BT_stack (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int ∗en, vrna_bp_stack_t ∗bp_stack, int ∗stack↩
  _count)

  *Backtrack a stacked pair closed by $(i, j)$.*
- int vrna_BT_int_loop (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int en, vrna_bp_stack_t ∗bp_stack, int
  ∗stack_count)

  *Backtrack an interior loop closed by $(i, j)$.*
- int vrna_E_mb_loop_stack (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate energy of a multi branch helices stacking onto closing pair (i,j)*
- int vrna_BT_mb_loop (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int ∗k, int en, int ∗component1, int
  ∗component2)

  *Backtrack the decomposition of a multi branch loop closed by $(i, j)$.*

## 15.2.2 Data Structure Documentation

### 15.2.2.1 struct vrna_mx_pf_aux_el_t

Auxiliary helper arrays for fast exterior loop computations.

**See also**

  vrna_exp_E_ext_fast_init(), vrna_exp_E_ext_fast_rotate(), vrna_exp_E_ext_fast_free(), vrna_exp_E_ext_↩
  fast()

### 15.2.2.2 struct vrna_mx_pf_aux_ml_t

Auxiliary helper arrays for fast exterior loop computations.

**See also**

  vrna_exp_E_ml_fast_init(), vrna_exp_E_ml_fast_rotate(), vrna_exp_E_ml_fast_free(), vrna_exp_E_ml_fast()

## 15.2.3 Function Documentation

### 15.2.3.1 vrna_E_ext_stem()

```
int vrna_E_ext_stem (
            unsigned int type,
            int n5d,
            int n3d,
            vrna_param_t * p )
```

#include <ViennaRNA/exterior_loops.h>

Evaluate a stem branching off the exterior loop.

Given a base pair $(i, j)$ encoded by *type*, compute the energy contribution including dangling-end/terminal-mismatch contributions. Instead of returning the energy contribution per-se, this function returns the corresponding Boltzmann factor. If either of the adjacent nucleotides $(i - 1)$ and $(j + 1)$ must not contribute stacking energy, the corresponding encoding must be $-1$.

**See also**

vrna_E_exp_stem()

**Parameters**

| type | The base pair encoding |
|------|------------------------|
| n5d  | The encoded nucleotide directly adjacent at the 5' side of the base pair (may be -1) |
| n3d  | The encoded nucleotide directly adjacent at the 3' side of the base pair (may be -1) |
| p    | The pre-computed energy parameters |

**Returns**

The energy contribution of the introduced exterior-loop stem

### 15.2.3.2 vrna_exp_E_ext_stem()

```
FLT_OR_DBL vrna_exp_E_ext_stem (
            unsigned int type,
            int n5d,
            int n3d,
            vrna_exp_param_t * p )
```

#include <ViennaRNA/exterior_loops.h>

Evaluate a stem branching off the exterior loop (Boltzmann factor version)

Given a base pair $(i, j)$ encoded by *type*, compute the energy contribution including dangling-end/terminal-mismatch contributions. Instead of returning the energy contribution per-se, this function returns the corresponding Boltzmann factor. If either of the adjacent nucleotides $(i - 1)$ and $(j + 1)$ must not contribute stacking energy, the corresponding encoding must be $-1$.

**See also**

vrna_E_ext_stem()

vrna_param_t * p )

**Parameters**

| type | The base pair encoding |
|---|---|
| n5d | The encoded nucleotide directly adjacent at the 5' side of the base pair (may be -1) |
| n3d | The encoded nucleotide directly adjacent at the 3' side of the base pair (may be -1) |
| p | The pre-computed energy parameters (Boltzmann factor version) |

**Returns**

> The Boltzmann weighted energy contribution of the introduced exterior-loop stem

### 15.2.3.3 E_Stem()

```
int E_Stem (
            int type,
            int si1,
            int sj1,
            int extLoop,
            vrna_param_t * P )
```

#include <ViennaRNA/exterior_loops.h>

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 behavior 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 incorporating 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 si1 or sj1 is a negative value, only 5' or 3' dangling end contributions are taken into account. To prohibit any of these mismatch contributions to be incorporated, 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 data structure containing scaled energy parameters |

**Returns**

>The Free energy of the branch off the loop in dcal/mol

**15.2.3.4  exp_E_ExtLoop()**

```
FLT_OR_DBL exp_E_ExtLoop (
            int type,
            int si1,
            int sj1,
            vrna_exp_param_t * P )
```

#include <ViennaRNA/exterior_loops.h>

This is the partition function variant of E_ExtLoop()

**Deprecated**  Use vrna_exp_E_ext_stem() instead!

**See also**

>E_ExtLoop()

**Returns**

>The Boltzmann weighted energy contribution of the introduced exterior-loop stem

**15.2.3.5  exp_E_Stem()**

```
FLT_OR_DBL exp_E_Stem (
            int type,
            int si1,
            int sj1,
            int extLoop,
            vrna_exp_param_t * P )
```

#include <ViennaRNA/exterior_loops.h>

**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**
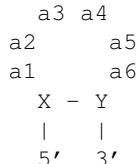
The Boltzmann weighted energy contribution of the branch off the loop

### 15.2.3.6    E_Hairpin()

```
PRIVATE int E_Hairpin (
            int size,
            int type,
            int si1,
            int sj1,
            const char * string,
            vrna_param_t * P )
```

#include <ViennaRNA/hairpin_loops.h>

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()
vrna_param_t

**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

**15.2.3.7 exp_E_Hairpin()**

```
PRIVATE FLT_OR_DBL exp_E_Hairpin (
            int u,
            int type,
            short si1,
            short sj1,
            const char * string,
            vrna_exp_param_t * P )
```

#include <ViennaRNA/hairpin_loops.h>

Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop.

multiply by scale[u+2]

**See also**

get_scaled_pf_parameters()
vrna_exp_param_t
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

### 15.2.3.8 vrna_E_hp_loop()

```
int vrna_E_hp_loop (
            vrna_fold_compound_t * vc,
            int i,
            int j )
```

#include <ViennaRNA/hairpin_loops.h>

Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.

This function evaluates the free energy of a hairpin loop

In case the base pair is not allowed due to a constraint conflict, this function returns INF.

**Note**

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_FC_TYPE_SINGLE or VRNA_FC_TYPE_COMPARATIVE

**Parameters**

| vc | The vrna_fold_compound_t that stores all relevant model settings |
|----|------------------------------------------------------------------|
| i | The 5' nucleotide of the base pair (3' to evaluate the pair as exterior hairpin loop) |
| j | The 3' nucleotide of the base pair (5' to evaluate the pair as exterior hairpin loop) |

**Returns**

The free energy of the hairpin loop in 10cal/mol

### 15.2.3.9 vrna_E_ext_hp_loop()

```
int vrna_E_ext_hp_loop (
            vrna_fold_compound_t * vc,
            int i,
            int j )
```

#include <ViennaRNA/hairpin_loops.h>

Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.

**Note**

This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_FC_TYPE_SINGLE or VRNA_FC_TYPE_COMPARATIVE

**15.2.3.10  vrna_eval_hp_loop()**

```
int vrna_eval_hp_loop (
            vrna_fold_compound_t * vc,
            int i,
            int j )
```

```
#include <ViennaRNA/hairpin_loops.h>
```

Evaluate free energy of a hairpin loop.

**Note**

> This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_FC_TYPE_SINGLE or VRNA_FC_TYPE_COMPARATIVE

**Parameters**

| | |
|---|---|
| vc | The vrna_fold_compound_t for the particular energy evaluation |
| i | 5'-position of the base pair |
| j | 3'-position of the base pair |

**Returns**

> Free energy of the hairpin loop closed by $(i, j)$ in deka-kal/mol

**SWIG Wrapper Notes**  This function is attached as method **eval_hp_loop()** to objects of type *fold_compound*

**15.2.3.11  vrna_exp_E_hp_loop()**

```
FLT_OR_DBL vrna_exp_E_hp_loop (
            vrna_fold_compound_t * vc,
            int i,
            int j )
```

```
#include <ViennaRNA/hairpin_loops.h>
```

High-Level function for hairpin loop energy evaluation (partition function variant)

**See also**

> vrna_E_hp_loop() for it's free energy counterpart

**Note**

> This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_FC_TYPE_SINGLE or VRNA_FC_TYPE_COMPARATIVE

**15.2.3.12   vrna_BT_hp_loop()**

```
int vrna_BT_hp_loop (
            vrna_fold_compound_t * vc,
            int i,
            int j,
            int en,
            vrna_bp_stack_t * bp_stack,
            int * stack_count )
```

```
#include <ViennaRNA/hairpin_loops.h>
```

Backtrack a hairpin loop closed by $(i, j)$.

**Note**

> This function is polymorphic! The provided vrna_fold_compound_t may be of type VRNA_FC_TYPE_SINGLE or VRNA_FC_TYPE_COMPARATIVE
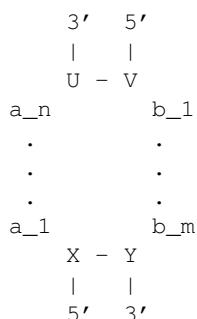
**15.2.3.13   E_IntLoop()**

```
PRIVATE int E_IntLoop (
            int n1,
            int n2,
            int type,
            int type_2,
            int si1,
            int sj1,
            int sp1,
            int sq1,
            vrna_param_t * P )
```

```
#include <ViennaRNA/interior_loops.h>
```

**Compute the Energy of an interior-loop**

This function computes the free energy $\Delta G$ 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()
> vrna_param_t

**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

**15.2.3.14 exp_E_IntLoop()**

```
PRIVATE FLT_OR_DBL exp_E_IntLoop (
            int u1,
            int u2,
            int type,
```

```
        int type2,
        short si1,
        short sj1,
        short sp1,
        short sq1,
        vrna_exp_param_t * P )
```

#include <ViennaRNA/interior_loops.h>

**Compute Boltzmann weight** $e^{-\Delta G/kT}$ **of interior loop**

multiply by scale[u1+u2+2] for scaling

**See also**

>   get_scaled_pf_parameters()
>   vrna_exp_param_t
>   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

### 15.2.3.15    vrna_eval_int_loop()

```
int vrna_eval_int_loop (
        vrna_fold_compound_t * vc,
        int i,
        int j,
        int k,
        int l )
```

#include <ViennaRNA/interior_loops.h>

Evaluate the free energy contribution of an interior loop with delimiting base pairs $(i, j)$ and $(k, l)$.

**Note**

> This function is polymorphic, i.e. it accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE as well
> as VRNA_FC_TYPE_COMPARATIVE

**SWIG Wrapper Notes**  This function is attached as method **eval_int_loop()** to objects of type *fold_compound*

**15.2.3.16  vrna_E_mb_loop_stack()**

```
int vrna_E_mb_loop_stack (
            vrna_fold_compound_t * vc,
            int i,
            int j )
```

`#include <ViennaRNA/multibranch_loops.h>`

Evaluate energy of a multi branch helices stacking onto closing pair (i,j)

Computes total free energy for coaxial stacking of (i.j) with (i+1.k) or (k+1.j-1)

**15.2.3.17  vrna_BT_mb_loop()**

```
int vrna_BT_mb_loop (
            vrna_fold_compound_t * vc,
            int * i,
            int * j,
            int * k,
            int en,
            int * component1,
            int * component2 )
```

`#include <ViennaRNA/multibranch_loops.h>`

Backtrack the decomposition of a multi branch loop closed by $(i, j)$.

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t filled with all relevant data for backtracking |
| *i* | 5' position of base pair closing the loop (will be set to 5' position of leftmost decomposed block upon successful backtracking) |
| *j* | 3' position of base pair closing the loop (will be set to 3' position of rightmost decomposed block upon successful backtracking) |
| *k* | Split position that delimits leftmost from rightmost block, [i,k] and [k+1, j], respectively. (Will be set upon successful backtracking) |
| *en* | The energy contribution of the substructure enclosed by $(i, j)$ |
| *component1* | Type of leftmost block (1 = ML, 2 = C) |
| *component2* | Type of rightmost block (1 = ML, 2 = C) |

**Returns**

1, if backtracking succeeded, 0 otherwise.

## 15.3 The RNA folding grammar

The RNA folding grammar as implemented in RNAlib.

### 15.3.1 Detailed Description

The RNA folding grammar as implemented in RNAlib.

Collaboration diagram for The RNA folding grammar:

### Modules

- Fine-tuning of the implemented models

  *Functions and data structures to fine-tune the implemented secondary structure evaluation model.*
- Energy parameters

  *All relevant functions to retrieve and copy pre-calculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).*
- Extending the folding grammar with additional domains

  *This module covers simple and straight-forward extensions to the RNA folding grammar.*
- Constraining the RNA folding grammar

  *This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.*

### Files

- file grammar.h

  *Implementations for the RNA folding grammar.*

### Data Structures

- struct vrna_gr_aux_s

### 15.3.2 Data Structure Documentation

#### 15.3.2.1 struct vrna_gr_aux_s

## 15.4 Fine-tuning of the implemented models

Functions and data structures to fine-tune the implemented secondary structure evaluation model.

### 15.4.1 Detailed Description

Functions and data structures to fine-tune the implemented secondary structure evaluation model.

Collaboration diagram for Fine-tuning of the implemented models:

### Files

- file model.h

  *The model details data structure and its corresponding modifiers.*

### Data Structures

- struct vrna_md_s

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

### Macros

- #define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0

  *Default temperature for structure prediction and free energy evaluation in $^{\circ}C$*
- #define VRNA_MODEL_DEFAULT_PF_SCALE -1

  *Default scaling factor for partition function computations.*
- #define VRNA_MODEL_DEFAULT_BETA_SCALE 1.

  *Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.*
- #define VRNA_MODEL_DEFAULT_DANGLES 2

  *Default dangling end model.*
- #define VRNA_MODEL_DEFAULT_SPECIAL_HP 1

  *Default model behavior for lookup of special tri-, tetra-, and hexa-loops.*
- #define VRNA_MODEL_DEFAULT_NO_LP 0

  *Default model behavior for so-called 'lonely pairs'.*
- #define VRNA_MODEL_DEFAULT_NO_GU 0

  *Default model behavior for G-U base pairs.*
- #define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0

  *Default model behavior for G-U base pairs closing a loop.*
- #define VRNA_MODEL_DEFAULT_CIRC 0

  *Default model behavior to treat a molecule as a circular RNA (DNA)*
- #define VRNA_MODEL_DEFAULT_GQUAD 0

  *Default model behavior regarding the treatment of G-Quadruplexes.*
- #define VRNA_MODEL_DEFAULT_UNIQ_ML 0

  *Default behavior of the model regarding unique multi-branch loop decomposition.*
- #define VRNA_MODEL_DEFAULT_ENERGY_SET 0

*Default model behavior on which energy set to use.*

- #define VRNA_MODEL_DEFAULT_BACKTRACK 1

  *Default model behavior with regards to backtracking of structures.*
- #define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'

  *Default model behavior on what type of backtracking to perform.*
- #define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1

  *Default model behavior with regards to computing base pair probabilities.*
- #define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1

  *Default model behavior for the allowed maximum base pair span.*
- #define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1

  *Default model behavior for the sliding window approach.*
- #define VRNA_MODEL_DEFAULT_LOG_ML 0

  *Default model behavior on how to evaluate the energy contribution of multi-branch loops.*
- #define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0

  *Default model behavior for consensus structure energy evaluation.*
- #define VRNA_MODEL_DEFAULT_ALI_RIBO 0

  *Default model behavior for consensus structure co-variance contribution assessment.*
- #define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.

  *Default model behavior for weighting the co-variance score in consensus structure prediction.*
- #define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

  *Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.*
- #define MAXALPHA 20

  *Maximal length of alphabet.*

## Typedefs

- typedef struct vrna_md_s vrna_md_t

  *Typename for the model details data structure vrna_md_s.*

## Functions

- void vrna_md_set_default (vrna_md_t ∗md)

  *Apply default model details to a provided vrna_md_t data structure.*
- void vrna_md_update (vrna_md_t ∗md)

  *Update the model details data structure.*
- vrna_md_t ∗ vrna_md_copy (vrna_md_t ∗md_to, const vrna_md_t ∗md_from)

  *Copy/Clone a vrna_md_t model.*
- char ∗ vrna_md_option_string (vrna_md_t ∗md)

  *Get a corresponding commandline parameter string of the options in a vrna_md_t.*
- void vrna_md_defaults_reset (vrna_md_t ∗md_p)

  *Reset the global default model details to a specific set of parameters, or their initial values.*
- void vrna_md_defaults_temperature (double T)

  *Set default temperature for energy evaluation of loops.*
- double vrna_md_defaults_temperature_get (void)

  *Get default temperature for energy evaluation of loops.*
- void vrna_md_defaults_betaScale (double b)

  *Set default scaling factor of thermodynamic temperature in Boltzmann factors.*
- double vrna_md_defaults_betaScale_get (void)

  *Get default scaling factor of thermodynamic temperature in Boltzmann factors.*

- void vrna_md_defaults_dangles (int d)

    *Set default dangle model for structure prediction.*
- int vrna_md_defaults_dangles_get (void)

    *Get default dangle model for structure prediction.*
- void vrna_md_defaults_special_hp (int flag)

    *Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*
- int vrna_md_defaults_special_hp_get (void)

    *Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*
- void vrna_md_defaults_noLP (int flag)

    *Set default behavior for prediction of canonical secondary structures.*
- int vrna_md_defaults_noLP_get (void)

    *Get default behavior for prediction of canonical secondary structures.*
- void vrna_md_defaults_noGU (int flag)

    *Set default behavior for treatment of G-U wobble pairs.*
- int vrna_md_defaults_noGU_get (void)

    *Get default behavior for treatment of G-U wobble pairs.*
- void vrna_md_defaults_noGUclosure (int flag)

    *Set default behavior for G-U pairs as closing pair for loops.*
- int vrna_md_defaults_noGUclosure_get (void)

    *Get default behavior for G-U pairs as closing pair for loops.*
- void vrna_md_defaults_logML (int flag)

    *Set default behavior recomputing free energies of multi-branch loops using a logarithmic model.*
- int vrna_md_defaults_logML_get (void)

    *Get default behavior recomputing free energies of multi-branch loops using a logarithmic model.*
- void vrna_md_defaults_circ (int flag)

    *Set default behavior whether input sequences are circularized.*
- int vrna_md_defaults_circ_get (void)

    *Get default behavior whether input sequences are circularized.*
- void vrna_md_defaults_gquad (int flag)

    *Set default behavior for treatment of G-Quadruplexes.*
- int vrna_md_defaults_gquad_get (void)

    *Get default behavior for treatment of G-Quadruplexes.*
- void vrna_md_defaults_uniq_ML (int flag)

    *Set default behavior for creating additional matrix for unique multi-branch loop prediction.*
- int vrna_md_defaults_uniq_ML_get (void)

    *Get default behavior for creating additional matrix for unique multi-branch loop prediction.*
- void vrna_md_defaults_energy_set (int e)

    *Set default energy set.*
- int vrna_md_defaults_energy_set_get (void)

    *Get default energy set.*
- void vrna_md_defaults_backtrack (int flag)

    *Set default behavior for whether to backtrack secondary structures.*
- int vrna_md_defaults_backtrack_get (void)

    *Get default behavior for whether to backtrack secondary structures.*
- void vrna_md_defaults_backtrack_type (char t)

    *Set default backtrack type, i.e. which DP matrix is used.*
- char vrna_md_defaults_backtrack_type_get (void)

    *Get default backtrack type, i.e. which DP matrix is used.*
- void vrna_md_defaults_compute_bpp (int flag)

    *Set the default behavior for whether to compute base pair probabilities after partition function computation.*
- int vrna_md_defaults_compute_bpp_get (void)

    *Get the default behavior for whether to compute base pair probabilities after partition function computation.*

- void vrna_md_defaults_max_bp_span (int span)

    *Set default maximal base pair span.*

- int vrna_md_defaults_max_bp_span_get (void)

    *Get default maximal base pair span.*

- void vrna_md_defaults_min_loop_size (int size)

    *Set default minimal loop size.*

- int vrna_md_defaults_min_loop_size_get (void)

    *Get default minimal loop size.*

- void vrna_md_defaults_window_size (int size)

    *Set default window size for sliding window structure prediction approaches.*

- int vrna_md_defaults_window_size_get (void)

    *Get default window size for sliding window structure prediction approaches.*

- void vrna_md_defaults_oldAliEn (int flag)

    *Set default behavior for whether to use old energy model for comparative structure prediction.*

- int vrna_md_defaults_oldAliEn_get (void)

    *Get default behavior for whether to use old energy model for comparative structure prediction.*

- void vrna_md_defaults_ribo (int flag)

    *Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.*

- int vrna_md_defaults_ribo_get (void)

    *Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.*

- void vrna_md_defaults_cv_fact (double factor)

    *Set the default co-variance scaling factor used in comparative structure prediction.*

- double vrna_md_defaults_cv_fact_get (void)

    *Get the default co-variance scaling factor used in comparative structure prediction.*

- void vrna_md_defaults_nc_fact (double factor)
- double vrna_md_defaults_nc_fact_get (void)
- void vrna_md_defaults_sfact (double factor)

    *Set the default scaling factor used to avoid under-/overflows in partition function computation.*

- double vrna_md_defaults_sfact_get (void)

    *Get the default scaling factor used to avoid under-/overflows in partition function computation.*

- void set_model_details (vrna_md_t ∗md)

    *Set default model details.*

## Variables

- double temperature

    *Rescale energy parameters to a temperature in degC.*

- double pf_scale

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

- int dangles

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

- int tetra_loop

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

- int noLonelyPairs

    *Global switch to avoid/allow helices of length 1.*

- 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 circ

    *backward compatibility variable.. this does not effect anything*

- int gquad

    *Allow G-quadruplex formation.*

- int uniq_ML

    *do ML decomposition uniquely (for subopt)*

- int energy_set

    *0 = BP; 1=any with GC; 2=any with AU-parameter*

- int do_backtrack

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

- char backtrack_type

    *A backtrack array marker for inverse_fold()*

- char ∗ nonstandards

    *contains allowed non standard base pairs*

- int max_bp_span

    *Maximum allowed base pair span.*

- int oldAliEn

    *use old alifold energies (with gaps)*

- int ribo

    *use ribosum matrices*

- int logML

    *if nonzero use logarithmic ML energy in energy_of_struct*

### 15.4.2 Data Structure Documentation

#### 15.4.2.1 struct vrna_md_s

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

For convenience reasons, we provide the type name vrna_md_t to address this data structure without the use of the struct keyword

**See also**

vrna_md_set_default(), set_model_details(), vrna_md_update(), vrna_md_t

**SWIG Wrapper Notes** This data structure is wrapped as an object **md** with multiple related functions attached as methods.

A new set of default parameters can be obtained by calling the constructure of **md:**

- *md()* – Initialize with default settings

The resulting object has a list of attached methods which directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *reset()* – vrna_md_set_default()
- *set_from_globals()* – set_model_details()
- *option_string()* – vrna_md_option_string()

Note, that default parameters can be modified by directly setting any of the following global variables. Internally, getting/setting default parameters using their global variable representative translates into calls of the following functions, therefore these wrappers for these functions do not exist in the scripting language interface(s):

| global variable | *C* getter | *C* setter |
|---|---|---|
| temperature | vrna_md_defaults_temperature_get() | vrna_md_defaults_temperature() |
| dangles | vrna_md_defaults_dangles_get() | vrna_md_defaults_dangles() |
| betaScale | vrna_md_defaults_betaScale_get() | vrna_md_defaults_betaScale() |
| tetra_loop | this is an alias of *special_hp* | |
| special_hp | vrna_md_defaults_special_hp_get() | vrna_md_defaults_special_hp() |
| noLonelyPairs | this is an alias of *noLP* | |
| noLP | vrna_md_defaults_noLP_get() | vrna_md_defaults_noLP() |
| noGU | vrna_md_defaults_noGU_get() | vrna_md_defaults_noGU() |
| no_closingGU | this is an alias of *noGUclosure* | |
| noGUclosure | vrna_md_defaults_noGUclosure_get() | vrna_md_defaults_noGUclosure() |
| logML | vrna_md_defaults_logML_get() | vrna_md_defaults_logML() |
| circ | vrna_md_defaults_circ_get() | vrna_md_defaults_circ() |
| gquad | vrna_md_defaults_gquad_get() | vrna_md_defaults_gquad() |
| uniq_ML | vrna_md_defaults_uniq_ML_get() | vrna_md_defaults_uniq_ML() |
| energy_set | vrna_md_defaults_energy_set_get() | vrna_md_defaults_energy_set() |
| backtrack | vrna_md_defaults_backtrack_get() | vrna_md_defaults_backtrack() |
| backtrack_type | vrna_md_defaults_backtrack_type_get() | vrna_md_defaults_backtrack_type() |
| do_backtrack | this is an alias of *compute_bpp* | |
| compute_bpp | vrna_md_defaults_compute_bpp_get() | vrna_md_defaults_compute_bpp() |
| max_bp_span | vrna_md_defaults_max_bp_span_get() | vrna_md_defaults_max_bp_span() |
| min_loop_size | vrna_md_defaults_min_loop_size_get() | vrna_md_defaults_min_loop_size() |
| window_size | vrna_md_defaults_window_size_get() | vrna_md_defaults_window_size() |
| oldAliEn | vrna_md_defaults_oldAliEn_get() | vrna_md_defaults_oldAliEn() |
| ribo | vrna_md_defaults_ribo_get() | vrna_md_defaults_ribo() |
| cv_fact | vrna_md_defaults_cv_fact_get() | vrna_md_defaults_cv_fact() |
| nc_fact | vrna_md_defaults_nc_fact_get() | vrna_md_defaults_nc_fact() |
| sfact | vrna_md_defaults_sfact_get() | vrna_md_defaults_sfact() |

**Data Fields**

- double temperature

  *The temperature used to scale the thermodynamic parameters.*
- double betaScale

  *A scaling factor for the thermodynamic temperature of the Boltzmann factors.*
- 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 multiloops.*

- int circ

  *Assume RNA to be circular instead of linear.*

- int gquad

  *Include G-quadruplexes in structure prediction.*

- int uniq_ML

  *Flag to ensure unique multi-branch loop decomposition during folding.*

- int energy_set

  *Specifies the energy set that defines set of compatible base pairs.*

- int backtrack

  *Specifies whether or not secondary structures should be backtraced.*

- char backtrack_type

  *Specifies in which matrix to backtrack.*

- int compute_bpp

  *Specifies whether or not backward recursions for base pair probability (bpp) computation will be performed.*

- char nonstandards [64]

  *contains allowed non standard bases*

- int max_bp_span

  *maximum allowed base pair span*

- int min_loop_size

  *Minimum size of hairpin loops.*

- int window_size

  *Size of the sliding window for locally optimal structure prediction.*

- int oldAliEn

  *Use old alifold energy model.*

- int ribo

  *Use ribosum scoring table in alifold energy model.*

- double cv_fact

  *Co-variance scaling factor for consensus structure prediction.*

- double nc_fact

  *Scaling factor to weight co-variance contributions of non-canonical pairs.*

- double sfact

  *Scaling factor for partition function scaling.*

- int rtype [8]

  *Reverse base pair type array.*

- short alias [MAXALPHA+1]

  *alias of an integer nucleotide representation*

- int pair [MAXALPHA+1][MAXALPHA+1]

  *Integer representation of a base pair.*

**15.4.2.1.1  Field Documentation**

**15.4.2.1.1.1 dangles**

`int vrna_md_s::dangles`

Specifies the dangle model used in any energy evaluation (0,1,2 or 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 vrna_pf() these checks are neglected. To provide comparability between free energy minimization and partition function algorithms, the default setting is 2. 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 set to 3 co-axial stacking is explicitly included for adjacent helices in multiloops. The option affects only mfe folding and energy evaluation (vrna_mfe() and vrna_eval_structure()), as well as suboptimal folding (vrna_subopt()) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

**Note**

> Some function do not implement all dangle model but only a subset of (0,1,2,3). In particular, partition function algorithms can only handle 0 and 2. Read the documentation of the particular recurrences or energy evaluation function for information about the provided dangle model.

**15.4.2.1.1.2 min_loop_size**

`int vrna_md_s::min_loop_size`

Minimum size of hairpin loops.

**Note**

> The default value for this field is TURN, however, it may be 0 in cofolding context.

## 15.4.3 Macro Definition Documentation

**15.4.3.1 VRNA_MODEL_DEFAULT_TEMPERATURE**

`#define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0`

`#include <ViennaRNA/model.h>`

Default temperature for structure prediction and free energy evaluation in $^{\circ}C$

**See also**

> vrna_md_t.temperature, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.2 VRNA_MODEL_DEFAULT_PF_SCALE**

`#define VRNA_MODEL_DEFAULT_PF_SCALE -1`

`#include <`ViennaRNA/model.h`>`

Default scaling factor for partition function computations.

**See also**

vrna_exp_param_t.pf_scale, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.3 VRNA_MODEL_DEFAULT_BETA_SCALE**

`#define VRNA_MODEL_DEFAULT_BETA_SCALE 1.`

`#include <`ViennaRNA/model.h`>`

Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.

**See also**

vrna_exp_param_t.alpha, vrna_md_t.betaScale, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.4 VRNA_MODEL_DEFAULT_DANGLES**

`#define VRNA_MODEL_DEFAULT_DANGLES 2`

`#include <`ViennaRNA/model.h`>`

Default dangling end model.

**See also**

vrna_md_t.dangles, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.5 VRNA_MODEL_DEFAULT_SPECIAL_HP**

`#define VRNA_MODEL_DEFAULT_SPECIAL_HP 1`

`#include <`ViennaRNA/model.h`>`

Default model behavior for lookup of special tri-, tetra-, and hexa-loops.

**See also**

vrna_md_t.special_hp, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.6  VRNA_MODEL_DEFAULT_NO_LP**

`#define VRNA_MODEL_DEFAULT_NO_LP 0`

`#include <`ViennaRNA/model.h`>`

Default model behavior for so-called 'lonely pairs'.

**See also**

vrna_md_t.noLP, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.7  VRNA_MODEL_DEFAULT_NO_GU**

`#define VRNA_MODEL_DEFAULT_NO_GU 0`

`#include <`ViennaRNA/model.h`>`

Default model behavior for G-U base pairs.

**See also**

vrna_md_t.noGU, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.8  VRNA_MODEL_DEFAULT_NO_GU_CLOSURE**

`#define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0`

`#include <`ViennaRNA/model.h`>`

Default model behavior for G-U base pairs closing a loop.

**See also**

vrna_md_t.noGUclosure, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.9  VRNA_MODEL_DEFAULT_CIRC**

`#define VRNA_MODEL_DEFAULT_CIRC 0`

`#include <`ViennaRNA/model.h`>`

Default model behavior to treat a molecule as a circular RNA (DNA)

**See also**

vrna_md_t.circ, vrna_md_defaults_reset(), vrna_md_set_default()

### 15.4.3.10 VRNA_MODEL_DEFAULT_GQUAD

#define VRNA_MODEL_DEFAULT_GQUAD 0

#include <ViennaRNA/model.h>

Default model behavior regarding the treatment of G-Quadruplexes.

**See also**

vrna_md_t.gquad, vrna_md_defaults_reset(), vrna_md_set_default()

### 15.4.3.11 VRNA_MODEL_DEFAULT_UNIQ_ML

#define VRNA_MODEL_DEFAULT_UNIQ_ML 0

#include <ViennaRNA/model.h>

Default behavior of the model regarding unique multi-branch loop decomposition.

**See also**

vrna_md_t.uniq_ML, vrna_md_defaults_reset(), vrna_md_set_default()

### 15.4.3.12 VRNA_MODEL_DEFAULT_ENERGY_SET

#define VRNA_MODEL_DEFAULT_ENERGY_SET 0

#include <ViennaRNA/model.h>

Default model behavior on which energy set to use.

**See also**

vrna_md_t.energy_set, vrna_md_defaults_reset(), vrna_md_set_default()

### 15.4.3.13 VRNA_MODEL_DEFAULT_BACKTRACK

#define VRNA_MODEL_DEFAULT_BACKTRACK 1

#include <ViennaRNA/model.h>

Default model behavior with regards to backtracking of structures.

**See also**

vrna_md_t.backtrack, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.14  VRNA_MODEL_DEFAULT_BACKTRACK_TYPE**

```
#define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'
```

```
#include <ViennaRNA/model.h>
```

Default model behavior on what type of backtracking to perform.

**See also**

vrna_md_t.backtrack_type, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.15  VRNA_MODEL_DEFAULT_COMPUTE_BPP**

```
#define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1
```

```
#include <ViennaRNA/model.h>
```

Default model behavior with regards to computing base pair probabilities.

**See also**

vrna_md_t.compute_bpp, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.16  VRNA_MODEL_DEFAULT_MAX_BP_SPAN**

```
#define VRNA_MODEL_DEFAULT_MAX_BP_SPAN −1
```

```
#include <ViennaRNA/model.h>
```

Default model behavior for the allowed maximum base pair span.

**See also**

vrna_md_t.max_bp_span, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.17  VRNA_MODEL_DEFAULT_WINDOW_SIZE**

```
#define VRNA_MODEL_DEFAULT_WINDOW_SIZE −1
```

```
#include <ViennaRNA/model.h>
```

Default model behavior for the sliding window approach.

**See also**

vrna_md_t.window_size, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.18   VRNA_MODEL_DEFAULT_LOG_ML**

```
#define VRNA_MODEL_DEFAULT_LOG_ML 0
```

`#include <`ViennaRNA/model.h`>`

Default model behavior on how to evaluate the energy contribution of multi-branch loops.

**See also**

vrna_md_t.logML, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.19   VRNA_MODEL_DEFAULT_ALI_OLD_EN**

```
#define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0
```

`#include <`ViennaRNA/model.h`>`

Default model behavior for consensus structure energy evaluation.

**See also**

vrna_md_t.oldAliEn, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.20   VRNA_MODEL_DEFAULT_ALI_RIBO**

```
#define VRNA_MODEL_DEFAULT_ALI_RIBO 0
```

`#include <`ViennaRNA/model.h`>`

Default model behavior for consensus structure co-variance contribution assessment.

**See also**

vrna_md_t.ribo, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.21   VRNA_MODEL_DEFAULT_ALI_CV_FACT**

```
#define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.
```

`#include <`ViennaRNA/model.h`>`

Default model behavior for weighting the co-variance score in consensus structure prediction.

**See also**

vrna_md_t.cv_fact, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.3.22 VRNA_MODEL_DEFAULT_ALI_NC_FACT**

#define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

#include <ViennaRNA/model.h>

Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.

**See also**

> vrna_md_t.nc_fact, vrna_md_defaults_reset(), vrna_md_set_default()

**15.4.4  Function Documentation**

**15.4.4.1  vrna_md_set_default()**

void vrna_md_set_default (
             vrna_md_t * md )

#include <ViennaRNA/model.h>

Apply default model details to a provided vrna_md_t data structure.

Use this function to initialize a vrna_md_t data structure with its default values

**Parameters**

| | |
|---|---|
| *md* | A pointer to the data structure that is about to be initialized |

**15.4.4.2  vrna_md_update()**

void vrna_md_update (
             vrna_md_t * md )

#include <ViennaRNA/model.h>

Update the model details data structure.

This function should be called after changing the vrna_md_t.energy_set attribute since it re-initializes base pairing related arrays within the vrna_md_t data structure. In particular, vrna_md_t.pair, vrna_md_t.alias, and vrna_md_t.rtype are set to the values that correspond to the specified vrna_md_t.energy_set option

**See also**

> vrna_md_t, vrna_md_t.energy_set, vrna_md_t.pair, vrna_md_t.rtype, vrna_md_t.alias, vrna_md_set_default()

**15.4.4.3 vrna_md_copy()**

[vrna_md_t]* vrna_md_copy (
        [vrna_md_t] * *md_to,*
        const [vrna_md_t] * *md_from* )

#include <[ViennaRNA/model.h]>

Copy/Clone a [vrna_md_t] model.

Use this function to clone a given model either inplace (target container `md_to` given) or create a copy by cloning the source model and returning it (`md_to == NULL`).

**Parameters**

| | |
|---|---|
| *md_to* | The model to be overwritten (if non-NULL and `md_to != md_from`) |
| *md_from* | The model to copy (if non-NULL) |

**Returns**

    A pointer to the copy model (or NULL if `md_from == NULL`)

**15.4.4.4 vrna_md_option_string()**

char* vrna_md_option_string (
        [vrna_md_t] * *md* )

#include <[ViennaRNA/model.h]>

Get a corresponding commandline parameter string of the options in a [vrna_md_t].

**Note**

    This function is not threadsafe!

**15.4.4.5 vrna_md_defaults_reset()**

void vrna_md_defaults_reset (
        [vrna_md_t] * *md_p* )

#include <[ViennaRNA/model.h]>

Reset the global default model details to a specific set of parameters, or their initial values.

This function resets the global default model details to their initial values, i.e. as specified by the ViennaRNA Package release, upon passing NULL as argument. Alternatively it resets them according to a set of provided parameters.

**Note**

>	The global default parameters affect all function calls of RNAlib where model details are not explicitly provided. Hence, any change of them is not considered threadsafe

**Warning**

>	This function first resets the global default settings to factory defaults, and only then applies user provided settings (if any). User settings that do not meet specifications are skipped.

**See also**

>	vrna_md_set_default(), vrna_md_t

**Parameters**

| md←_p | A set of model details to use as global default (if NULL is passed, factory defaults are restored) |
|---|---|

**15.4.4.6   vrna_md_defaults_temperature()**

```
void vrna_md_defaults_temperature (
            double T )
```

#include <ViennaRNA/model.h>

Set default temperature for energy evaluation of loops.

**See also**

>	vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_TEMPERATURE

**Parameters**

| T | Temperature in centigrade |
|---|---|

**15.4.4.7   vrna_md_defaults_temperature_get()**

```
double vrna_md_defaults_temperature_get (
            void  )
```

#include <ViennaRNA/model.h>

Get default temperature for energy evaluation of loops.

**See also**

vrna_md_defaults_temperature(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_T

**Returns**

The global default settings for temperature in centigrade

**15.4.4.8 vrna_md_defaults_betaScale()**

```
void vrna_md_defaults_betaScale (
            double b )
```

#include <ViennaRNA/model.h>

Set default scaling factor of thermodynamic temperature in Boltzmann factors.

Bolzmann factors are then computed as $exp(-E/(b \cdot kT))$.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BETA_SCALE

**Parameters**

| b | The scaling factor, default is 1.0 |
|---|---|

**15.4.4.9 vrna_md_defaults_betaScale_get()**

```
double vrna_md_defaults_betaScale_get (
            void  )
```

#include <ViennaRNA/model.h>

Get default scaling factor of thermodynamic temperature in Boltzmann factors.

**See also**

vrna_md_defaults_betaScale(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BET

**Returns**

The global default thermodynamic temperature scaling factor

**15.4.4.10 vrna_md_defaults_dangles()**

```
void vrna_md_defaults_dangles (
            int d )
```

#include <ViennaRNA/model.h>

Set default dangle model for structure prediction.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_DANGLES

**Parameters**

| d | The dangle model |
|---|---|

**15.4.4.11 vrna_md_defaults_dangles_get()**

```
int vrna_md_defaults_dangles_get (
            void )
```

#include <ViennaRNA/model.h>

Get default dangle model for structure prediction.

**See also**

vrna_md_defaults_dangles(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_DANG

**Returns**

The global default settings for the dangle model

**15.4.4.12 vrna_md_defaults_special_hp()**

```
void vrna_md_defaults_special_hp (
            int flag )
```

#include <ViennaRNA/model.h>

Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_SPECIAL_HP

**Parameters**

| | |
|---|---|
| *flag* | On/Off switch (0 = OFF, else = ON) |

**15.4.4.13 vrna_md_defaults_special_hp_get()**

```
int vrna_md_defaults_special_hp_get (
             void )
```

#include <ViennaRNA/model.h>

Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

**See also**

vrna_md_defaults_special_hp(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_SP

**Returns**

The global default settings for the treatment of special hairpin loops

**15.4.4.14 vrna_md_defaults_noLP()**

```
void vrna_md_defaults_noLP (
             int flag )
```

#include <ViennaRNA/model.h>

Set default behavior for prediction of canonical secondary structures.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_LP

**Parameters**

| | |
|---|---|
| *flag* | On/Off switch (0 = OFF, else = ON) |

**15.4.4.15 vrna_md_defaults_noLP_get()**

```
int vrna_md_defaults_noLP_get (
```

```
        void  )
```

#include <ViennaRNA/model.h>

Get default behavior for prediction of canonical secondary structures.

**See also**

vrna_md_defaults_noLP(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_LP

**Returns**

The global default settings for predicting canonical secondary structures

### 15.4.4.16   vrna_md_defaults_noGU()

```
void vrna_md_defaults_noGU (
            int flag )
```

#include <ViennaRNA/model.h>

Set default behavior for treatment of G-U wobble pairs.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|------|-------------------------------------|

### 15.4.4.17   vrna_md_defaults_noGU_get()

```
int vrna_md_defaults_noGU_get (
            void  )
```

#include <ViennaRNA/model.h>

Get default behavior for treatment of G-U wobble pairs.

**See also**

vrna_md_defaults_noGU(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU

**Returns**

The global default settings for treatment of G-U wobble pairs

**15.4.4.18 vrna_md_defaults_noGUclosure()**

```
void vrna_md_defaults_noGUclosure (
            int flag )
```

#include <ViennaRNA/model.h>

Set default behavior for G-U pairs as closing pair for loops.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_NO_GU_CLOSURE

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|------|-------------------------------------|

**15.4.4.19 vrna_md_defaults_noGUclosure_get()**

```
int vrna_md_defaults_noGUclosure_get (
            void  )
```

#include <ViennaRNA/model.h>

Get default behavior for G-U pairs as closing pair for loops.

**See also**

vrna_md_defaults_noGUclosure(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_N

**Returns**

The global default settings for treatment of G-U pairs closing a loop

**15.4.4.20 vrna_md_defaults_logML()**

```
void vrna_md_defaults_logML (
            int flag )
```

#include <ViennaRNA/model.h>

Set default behavior recomputing free energies of multi-branch loops using a logarithmic model.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_LOG_ML

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|------|-------------------------------------|

**15.4.4.21 vrna_md_defaults_logML_get()**

```
int vrna_md_defaults_logML_get (
              void  )
```

#include <ViennaRNA/model.h>

Get default behavior recomputing free energies of multi-branch loops using a logarithmic model.

**See also**

vrna_md_defaults_logML(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_LOG_M

**Returns**

The global default settings for logarithmic model in multi-branch loop free energy evaluation

**15.4.4.22 vrna_md_defaults_circ()**

```
void vrna_md_defaults_circ (
              int flag )
```

#include <ViennaRNA/model.h>

Set default behavior whether input sequences are circularized.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_CIRC

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|------|-------------------------------------|

**15.4.4.23 vrna_md_defaults_circ_get()**

```
int vrna_md_defaults_circ_get (
              void  )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior whether input sequences are circularized.

**See also**

vrna_md_defaults_circ(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_CIRC

**Returns**

The global default settings for treating input sequences as circular

**15.4.4.24    vrna_md_defaults_gquad()**

```
void vrna_md_defaults_gquad (
            int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for treatment of G-Quadruplexes.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_GQUAD

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
| --- | --- |

**15.4.4.25    vrna_md_defaults_gquad_get()**

```
int vrna_md_defaults_gquad_get (
            void  )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for treatment of G-Quadruplexes.

**See also**

vrna_md_defaults_gquad(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_GQUAD

**Returns**

The global default settings for treatment of G-Quadruplexes

**15.4.4.26 vrna_md_defaults_uniq_ML()**

```
void vrna_md_defaults_uniq_ML (
            int flag )
```

#include <ViennaRNA/model.h>

Set default behavior for creating additional matrix for unique multi-branch loop prediction.

**Note**

Activating this option usually results in higher memory consumption!

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_UNIQ_ML

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|------|-----------------------------------|

**15.4.4.27 vrna_md_defaults_uniq_ML_get()**

```
int vrna_md_defaults_uniq_ML_get (
            void )
```

#include <ViennaRNA/model.h>

Get default behavior for creating additional matrix for unique multi-branch loop prediction.

**See also**

vrna_md_defaults_uniq_ML(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_UNIQ

**Returns**

The global default settings for creating additional matrices for unique multi-branch loop prediction

**15.4.4.28 vrna_md_defaults_energy_set()**

```
void vrna_md_defaults_energy_set (
            int e )
```

#include <ViennaRNA/model.h>

Set default energy set.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ENERGY_SET

**Parameters**

| e | Energy set (0, 1, 2, 3) |
|---|---|

**15.4.4.29    vrna_md_defaults_energy_set_get()**

```
int vrna_md_defaults_energy_set_get (
              void  )
```

#include <ViennaRNA/model.h>

Get default energy set.

**See also**

vrna_md_defaults_energy_set(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_EN

**Returns**

The global default settings for the energy set

**15.4.4.30    vrna_md_defaults_backtrack()**

```
void vrna_md_defaults_backtrack (
              int  flag )
```

#include <ViennaRNA/model.h>

Set default behavior for whether to backtrack secondary structures.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BACKTRACK

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|---|---|

**15.4.4.31    vrna_md_defaults_backtrack_get()**

```
int vrna_md_defaults_backtrack_get (
              void  )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to backtrack secondary structures.

**See also**

vrna_md_defaults_backtrack(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BAC

**Returns**

The global default settings for backtracking structures

**15.4.4.32 vrna_md_defaults_backtrack_type()**

```
void vrna_md_defaults_backtrack_type (
            char t )
```

```
#include <ViennaRNA/model.h>
```

Set default backtrack type, i.e. which DP matrix is used.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_BACKTRACK_TYPE

**Parameters**

| t | The type ('F', 'C', or 'M') |
|---|---|

**15.4.4.33 vrna_md_defaults_backtrack_type_get()**

```
char vrna_md_defaults_backtrack_type_get (
            void  )
```

```
#include <ViennaRNA/model.h>
```

Get default backtrack type, i.e. which DP matrix is used.

**See also**

vrna_md_defaults_backtrack_type(),   vrna_md_defaults_reset(),   vrna_md_set_default(),   vrna_md_t,
VRNA_MODEL_DEFAULT_BACKTRACK_TYPE

**Returns**

The global default settings that specify which DP matrix is used for backtracking

**15.4.4.34    vrna_md_defaults_compute_bpp()**

```
void vrna_md_defaults_compute_bpp (
            int flag )
```

#include <ViennaRNA/model.h>

Set the default behavior for whether to compute base pair probabilities after partition function computation.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_COMPUTE_BPP

**Parameters**

| flag | On/Off switch (0 = OFF, else = ON) |
|------|-------------------------------------|

**15.4.4.35    vrna_md_defaults_compute_bpp_get()**

```
int vrna_md_defaults_compute_bpp_get (
            void  )
```

#include <ViennaRNA/model.h>

Get the default behavior for whether to compute base pair probabilities after partition function computation.

**See also**

vrna_md_defaults_compute_bpp(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_

**Returns**

The global default settings that specify whether base pair probabilities are computed together with partition function

**15.4.4.36    vrna_md_defaults_max_bp_span()**

```
void vrna_md_defaults_max_bp_span (
            int span )
```

#include <ViennaRNA/model.h>

Set default maximal base pair span.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_MAX_BP_SPAN

**Parameters**

| | |
|---|---|
| *span* | Maximal base pair span |

**15.4.4.37 vrna_md_defaults_max_bp_span_get()**

```
int vrna_md_defaults_max_bp_span_get (
            void  )
```

`#include <`[`ViennaRNA/model.h`](#)`>`

Get default maximal base pair span.

**See also**

> [vrna_md_defaults_max_bp_span()](#), [vrna_md_defaults_reset()](#), [vrna_md_set_default()](#), [vrna_md_t](#), [VRNA_MODEL_DEFAULT_](#)

**Returns**

> The global default settings for maximum base pair span

**15.4.4.38 vrna_md_defaults_min_loop_size()**

```
void vrna_md_defaults_min_loop_size (
            int size )
```

`#include <`[`ViennaRNA/model.h`](#)`>`

Set default minimal loop size.

**See also**

> [vrna_md_defaults_reset()](#), [vrna_md_set_default()](#), [vrna_md_t](#), [TURN](#)

**Parameters**

| | |
|---|---|
| *size* | Minimal size, i.e. number of unpaired nucleotides for a hairpin loop |

**15.4.4.39 vrna_md_defaults_min_loop_size_get()**

```
int vrna_md_defaults_min_loop_size_get (
            void  )
```

```
#include <ViennaRNA/model.h>
```

Get default minimal loop size.

**See also**

vrna_md_defaults_min_loop_size(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, TURN

**Returns**

The global default settings for minimal size of hairpin loops

**15.4.4.40   vrna_md_defaults_window_size()**

```
void vrna_md_defaults_window_size (
            int size )
```

```
#include <ViennaRNA/model.h>
```

Set default window size for sliding window structure prediction approaches.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_WINDOW_SIZE

**Parameters**

| size | The size of the sliding window |
| --- | --- |

**15.4.4.41   vrna_md_defaults_window_size_get()**

```
int vrna_md_defaults_window_size_get (
            void  )
```

```
#include <ViennaRNA/model.h>
```

Get default window size for sliding window structure prediction approaches.

**See also**

vrna_md_defaults_window_size(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_V

**Returns**

The global default settings for the size of the sliding window

**15.4.4.42 vrna_md_defaults_oldAliEn()**

```
void vrna_md_defaults_oldAliEn (
            int flag )
```

#include <[ViennaRNA/model.h](#)>

Set default behavior for whether to use old energy model for comparative structure prediction.

**Note**

This option is outdated. Activating the old energy model usually results in worse consensus structure predictions.

**See also**

[vrna_md_defaults_reset()](#), [vrna_md_set_default()](#), [vrna_md_t](#), [VRNA_MODEL_DEFAULT_ALI_OLD_EN](#)

**Parameters**

| | |
|---|---|
| *flag* | On/Off switch (0 = OFF, else = ON) |

**15.4.4.43 vrna_md_defaults_oldAliEn_get()**

```
int vrna_md_defaults_oldAliEn_get (
            void )
```

#include <[ViennaRNA/model.h](#)>

Get default behavior for whether to use old energy model for comparative structure prediction.

**See also**

[vrna_md_defaults_oldAliEn()](#), [vrna_md_defaults_reset()](#), [vrna_md_set_default()](#), [vrna_md_t](#), [VRNA_MODEL_DEFAULT_ALI_(](#)

**Returns**

The global default settings for using old energy model for comparative structure prediction

**15.4.4.44 vrna_md_defaults_ribo()**

```
void vrna_md_defaults_ribo (
            int flag )
```

#include <[ViennaRNA/model.h](#)>

Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.

**See also**

[vrna_md_defaults_reset()](#), [vrna_md_set_default()](#), [vrna_md_t](#), [VRNA_MODEL_DEFAULT_ALI_RIBO](#)

**Parameters**

| *flag* | On/Off switch (0 = OFF, else = ON) |

**15.4.4.45  vrna_md_defaults_ribo_get()**

```
int vrna_md_defaults_ribo_get (
            void  )
```

`#include <`ViennaRNA/model.h`>`

Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.

**See also**

> vrna_md_defaults_ribo(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_RIBO

**Returns**

> The global default settings for using Ribosum scoring in comparative structure prediction

**15.4.4.46  vrna_md_defaults_cv_fact()**

```
void vrna_md_defaults_cv_fact (
            double factor )
```

`#include <`ViennaRNA/model.h`>`

Set the default co-variance scaling factor used in comparative structure prediction.

**See also**

> vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_CV_FACT

**Parameters**

| *factor* | The co-variance factor |

**15.4.4.47  vrna_md_defaults_cv_fact_get()**

```
double vrna_md_defaults_cv_fact_get (
            void  )
```

```
#include <ViennaRNA/model.h>
```

Get the default co-variance scaling factor used in comparative structure prediction.

**See also**

> vrna_md_defaults_cv_fact(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_CV

**Returns**

> The global default settings for the co-variance factor

**15.4.4.48 vrna_md_defaults_nc_fact()**

```
void vrna_md_defaults_nc_fact (
            double factor )
```

```
#include <ViennaRNA/model.h>
```

**See also**

> vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_NC_FACT

**Parameters**

| factor | |
|--------|--|

**15.4.4.49 vrna_md_defaults_nc_fact_get()**

```
double vrna_md_defaults_nc_fact_get (
            void  )
```

```
#include <ViennaRNA/model.h>
```

**See also**

> vrna_md_defaults_nc_fact(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t, VRNA_MODEL_DEFAULT_ALI_N

**Returns**

**15.4.4.50   vrna_md_defaults_sfact()**

```
void vrna_md_defaults_sfact (
            double factor )
```

#include <ViennaRNA/model.h>

Set the default scaling factor used to avoid under-/overflows in partition function computation.

**See also**

vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t

**Parameters**

| | |
|---|---|
| *factor* | The scaling factor (default: 1.07) |

**15.4.4.51   vrna_md_defaults_sfact_get()**

```
double vrna_md_defaults_sfact_get (
            void  )
```

#include <ViennaRNA/model.h>

Get the default scaling factor used to avoid under-/overflows in partition function computation.

**See also**

vrna_md_defaults_sfact(), vrna_md_defaults_reset(), vrna_md_set_default(), vrna_md_t

**Returns**

The global default settings of the scaling factor

**15.4.4.52   set_model_details()**

```
void set_model_details (
            vrna_md_t * md )
```

#include <ViennaRNA/model.h>

Set default model details.

Use this function if you wish to initialize a vrna_md_t data structure with its default values, i.e. the global model settings as provided by the deprecated global variables.

**Deprecated** This function will vanish as soon as backward compatibility of RNAlib is dropped (expected in version 3). Use vrna_md_set_default() instead!

**Parameters**

| *md* | A pointer to the data structure that is about to be initialized |
|------|----------------------------------------------------------------|

### 15.4.5 Variable Documentation

#### 15.4.5.1 temperature

```
double temperature
```

```
#include <ViennaRNA/model.h>
```

Rescale energy parameters to a temperature in degC.

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

**Deprecated** Use vrna_md_defaults_temperature(), and vrna_md_defaults_temperature_get() to change, and read the global default temperature settings

**See also**

vrna_md_defaults_temperature(), vrna_md_defaults_temperature_get(), vrna_md_defaults_reset()

#### 15.4.5.2 pf_scale

```
double pf_scale
```

```
#include <ViennaRNA/model.h>
```

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.

### 15.4.5.3 dangles

`int dangles`

`#include <ViennaRNA/model.h>`

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 multiloops. 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

### 15.4.5.4 tetra_loop

`int tetra_loop`

`#include <ViennaRNA/model.h>`

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

default is 1.

### 15.4.5.5 noLonelyPairs

`int noLonelyPairs`

`#include <ViennaRNA/model.h>`

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.

### 15.4.5.6 energy_set

`int energy_set`

`#include <ViennaRNA/model.h>`

0 = BP; 1=any with GC; 2=any with 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.

**15.4.5.7 do_backtrack**

```
int do_backtrack
```

#include <ViennaRNA/model.h>

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.

**15.4.5.8 backtrack_type**

```
char backtrack_type
```

#include <ViennaRNA/model.h>

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 multiloop. Otherwise ('F') the usual mfe structure is computed.

**15.4.5.9 nonstandards**

```
char* nonstandards
```

#include <ViennaRNA/model.h>

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.

**15.4.5.10 max_bp_span**

```
int max_bp_span
```

#include <ViennaRNA/model.h>

Maximum allowed base pair span.

A value of -1 indicates no restriction for distant base pairs.

## 15.5 Energy parameters

All relevant functions to retrieve and copy pre-calculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

### 15.5.1 Detailed Description

All relevant functions to retrieve and copy pre-calculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

This module covers all relevant functions for pre-calculation 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. Collaboration diagram for Energy parameters:

### Modules

- Reading/Writing Energy Parameter Sets from/to File

    *Read and Write energy parameter sets from and to text files.*

### Files

- file convert_epars.h

    *Functions and definitions for energy parameter file format conversion.*
- file energy_const.h

    *Energy parameter constants.*
- file params.h

    *Functions to deal with sets of energy parameters.*
- file read_epars.h

    *Read and write energy parameter files.*

### Data Structures

- struct vrna_param_s

    *The datastructure that contains temperature scaled energy parameters. More...*
- struct vrna_exp_param_s

    *The data structure that contains temperature scaled Boltzmann weights of the energy parameters. More...*

### Typedefs

- typedef struct vrna_param_s vrna_param_t

    *Typename for the free energy parameter data structure vrna_params.*
- typedef struct vrna_exp_param_s vrna_exp_param_t

    *Typename for the Boltzmann factor data structure vrna_exp_params.*
- typedef struct vrna_param_s paramT

    *Old typename of vrna_param_s.*
- typedef struct vrna_exp_param_s pf_paramT

    *Old typename of vrna_exp_param_s.*

**Functions**

- vrna_param_t ∗ vrna_params (vrna_md_t ∗md)

  *Get a data structure containing prescaled free energy parameters.*

- vrna_param_t ∗ vrna_params_copy (vrna_param_t ∗par)

  *Get a copy of the provided free energy parameters.*

- vrna_exp_param_t ∗ vrna_exp_params (vrna_md_t ∗md)

  *Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.*

- vrna_exp_param_t ∗ vrna_exp_params_comparative (unsigned int n_seq, vrna_md_t ∗md)

  *Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)*

- vrna_exp_param_t ∗ vrna_exp_params_copy (vrna_exp_param_t ∗par)

  *Get a copy of the provided free energy parameters (provided as Boltzmann factors)*

- void vrna_params_subst (vrna_fold_compound_t ∗vc, vrna_param_t ∗par)

  *Update/Reset energy parameters data structure within a vrna_fold_compound_t.*

- void vrna_exp_params_subst (vrna_fold_compound_t ∗vc, vrna_exp_param_t ∗params)

  *Update the energy parameters for subsequent partition function computations.*

- void vrna_exp_params_rescale (vrna_fold_compound_t ∗vc, double ∗mfe)

  *Rescale Boltzmann factors for partition function computations.*

- void vrna_params_reset (vrna_fold_compound_t ∗vc, vrna_md_t ∗md_p)

  *Reset free energy parameters within a vrna_fold_compound_t according to provided, or default model details.*

- void vrna_exp_params_reset (vrna_fold_compound_t ∗vc, vrna_md_t ∗md_p)

  *Reset Boltzmann factors for partition function computations within a vrna_fold_compound_t according to provided, or default model details.*

- vrna_exp_param_t ∗ get_scaled_pf_parameters (void)

- vrna_exp_param_t ∗ get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)

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

- vrna_exp_param_t ∗ get_boltzmann_factor_copy (vrna_exp_param_t ∗parameters)

  *Get a copy of already precomputed Boltzmann factors.*

- vrna_exp_param_t ∗ get_scaled_alipf_parameters (unsigned int n_seq)

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

- vrna_exp_param_t ∗ get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)

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

- vrna_param_t ∗ scale_parameters (void)

  *Get precomputed energy contributions for all the known loop types.*

- vrna_param_t ∗ get_scaled_parameters (double temperature, vrna_md_t md)

  *Get precomputed energy contributions for all the known loop types.*

## 15.5.2 Data Structure Documentation

### 15.5.2.1 struct vrna_param_s

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for vrna_param_s:

**Data Fields**

- double temperature

    *Temperature used for loop contribution scaling.*
- vrna_md_t model_details

    *Model details to be used in the recursions.*
- char param_file [256]

    *The filename the parameters were derived from, or empty string if they represent the default.*

**15.5.2.2   struct vrna_exp_param_s**

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

Collaboration diagram for vrna_exp_param_s:

**Data Fields**

- int id

    *An identifier for the data structure.*
- 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.*
- vrna_md_t model_details

    *Model details to be used in the recursions.*
- char param_file [256]

    *The filename the parameters were derived from, or empty string if they represent the default.*

**15.5.2.2.1   Field Documentation**

**15.5.2.2.1.1   id**

```
int vrna_exp_param_s::id
```

An identifier for the data structure.

**Deprecated**   This attribute will be removed in version 3

**15.5.2.2.1.2 alpha**

```
double vrna_exp_param_s::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 $e^{-E/(\alpha \cdot K \cdot T)}$

**15.5.3 Typedef Documentation**

**15.5.3.1 paramT**

```
typedef struct vrna_param_s paramT
```

`#include <ViennaRNA/params.h>`

Old typename of vrna_param_s.

**Deprecated** Use vrna_param_t instead!

**15.5.3.2 pf_paramT**

```
typedef struct vrna_exp_param_s pf_paramT
```

`#include <ViennaRNA/params.h>`

Old typename of vrna_exp_param_s.

**Deprecated** Use vrna_exp_param_t instead!

**15.5.4 Function Documentation**

**15.5.4.1 vrna_params()**

```
vrna_param_t* vrna_params (
              vrna_md_t * md )
```

`#include <ViennaRNA/params.h>`

Get a data structure containing prescaled free energy parameters.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested vrna_param_t structure.

**See also**

vrna_md_t, vrna_md_set_default(), vrna_exp_params()

**Parameters**

| md | A pointer to the model details to store inside the structure (Maybe NULL) |
|---|---|

**Returns**

A pointer to the memory location where the requested parameters are stored

### 15.5.4.2 vrna_params_copy()

vrna_param_t* vrna_params_copy (
            vrna_param_t * *par* )

#include <ViennaRNA/params.h>

Get a copy of the provided free energy parameters.

If NULL is passed as parameter, a default set of energy parameters is created and returned.

**See also**

vrna_params(), vrna_param_t

**Parameters**

| par | The free energy parameters that are to be copied (Maybe NULL) |
|---|---|

**Returns**

A copy or a default set of the (provided) parameters

### 15.5.4.3 vrna_exp_params()

vrna_exp_param_t* vrna_exp_params (
            vrna_md_t * *md* )

#include <ViennaRNA/params.h>

Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.

This function returns a data structure that contains all necessary precomputed energy contributions for each type of loop.

In contrast to vrna_params(), the free energies within this data structure are stored as their Boltzmann factors, i.e.

$exp(-E/kT)$

where $E$ is the free energy.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested vrna_exp_param_t structure.

**See also**

vrna_md_t, vrna_md_set_default(), vrna_params(), vrna_rescale_pf_params()

**Parameters**

| | |
|---|---|
| *md* | A pointer to the model details to store inside the structure (Maybe NULL) |

**Returns**

A pointer to the memory location where the requested parameters are stored

**15.5.4.4 vrna_exp_params_comparative()**

```
vrna_exp_param_t* vrna_exp_params_comparative (
            unsigned int n_seq,
            vrna_md_t * md )
```

```
#include <ViennaRNA/params.h>
```

Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested vrna_exp_param_t structure.

**See also**

vrna_md_t, vrna_md_set_default(), vrna_exp_params(), vrna_params()

**Parameters**

| | |
|---|---|
| *n_seq* | The number of sequences in the alignment |
| *md* | A pointer to the model details to store inside the structure (Maybe NULL) |

**Returns**

A pointer to the memory location where the requested parameters are stored

**15.5.4.5 vrna_exp_params_copy()**

```
vrna_exp_param_t* vrna_exp_params_copy (
            vrna_exp_param_t * par )
```

```
#include <ViennaRNA/params.h>
```

Get a copy of the provided free energy parameters (provided as Boltzmann factors)

If NULL is passed as parameter, a default set of energy parameters is created and returned.

**See also**

[vrna_exp_params()](), [vrna_exp_param_t]()

**Parameters**

| | |
|---|---|
| *par* | The free energy parameters that are to be copied (Maybe NULL) |

**Returns**

A copy or a default set of the (provided) parameters

**15.5.4.6   vrna_params_subst()**

```
void vrna_params_subst (
            vrna_fold_compound_t * vc,
            vrna_param_t * par )
```

`#include <ViennaRNA/params.h>`

Update/Reset energy parameters data structure within a [vrna_fold_compound_t]().

Passing NULL as second argument leads to a reset of the energy parameters within vc to their default values. Otherwise, the energy parameters provided will be copied over into vc.

**See also**

[vrna_params_reset()](), [vrna_param_t](), [vrna_md_t](), [vrna_params()]()

**Parameters**

| | |
|---|---|
| *vc* | The [vrna_fold_compound_t]() that is about to receive updated energy parameters |
| *par* | The energy parameters used to substitute those within vc (Maybe NULL) |

**SWIG Wrapper Notes**   This function is attached to [vrna_fc_s]() objects as **params_subst()** method.

**15.5.4.7   vrna_exp_params_subst()**

```
void vrna_exp_params_subst (
            vrna_fold_compound_t * vc,
            vrna_exp_param_t * params )
```

`#include <ViennaRNA/params.h>`

Update the energy parameters for subsequent partition function computations.

This function can be used to properly assign new energy parameters for partition function computations to a vrna_fold_compound_t. For this purpose, the data of the provided pointer `params` will be copied into `vc` and a recomputation of the partition function scaling factor is issued, if the `pf_scale` attribute of `params` is less than `1.0`.

Passing NULL as second argument leads to a reset of the energy parameters within vc to their default values

**See also**

> vrna_exp_params_reset(), vrna_exp_params_rescale(), vrna_exp_param_t, vrna_md_t, vrna_exp_params()

**Parameters**

| vc | The fold compound data structure |
|---|---|
| params | A pointer to the new energy parameters |

**15.5.4.8   vrna_exp_params_rescale()**

```
void vrna_exp_params_rescale (
            vrna_fold_compound_t * vc,
            double * mfe )
```

`#include <ViennaRNA/params.h>`

Rescale Boltzmann factors for partition function computations.

This function may be used to (automatically) rescale the Boltzmann factors used in partition function computations. Since partition functions over subsequences can easily become extremely large, the RNAlib internally rescales them to avoid numerical over- and/or underflow. Therefore, a proper scaling factor $s$ needs to be chosen that in turn is then used to normalize the corresponding partition functions $\hat{q}[i,j] = q[i,j]/s^{(j-i+1)}$.

This function provides two ways to automatically adjust the scaling factor.

1. Automatic guess

2. Automatic adjustment according to MFE

Passing `NULL` as second parameter activates the *automatic guess mode*. Here, the scaling factor is recomputed according to a mean free energy of `184.3*length` cal for random sequences.

**Note**

> This recomputation only takes place if the `pf_scale` attribute of the `exp_params` data structure contained in `vc` has a value below `1.0`.

On the other hand, if the MFE for a sequence is known, it can be used to recompute a more robust scaling factor, since it represents the lowest free energy of the entire ensemble of structures, i.e. the highest Boltzmann factor. To activate this second mode of *automatic adjustment according to MFE*, a pointer to the MFE value needs to be passed as second argument. This value is then taken to compute the scaling factor as $s = exp((sfact * MFE)/kT/length)$, where sfact is an additional scaling weight located in the vrna_md_t data structure of $exp\hookleftarrow$ _params in vc.

The computed scaling factor $s$ will be stored as `pf_scale` attribute of the `exp_params` data structure in `vc`.

**See also**

> vrna_exp_params_subst(), vrna_md_t, vrna_exp_param_t, vrna_fold_compound_t

**Parameters**

| | |
|---|---|
| *vc* | The fold compound data structure |
| *mfe* | A pointer to the MFE (in kcal/mol) or NULL |

**SWIG Wrapper Notes** This function is attached to vrna_fc_s objects as overloaded **exp_params_rescale()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_rescale(), i.e. default scaling of the partition function. Passing an energy in kcal/mol, e.g. as retrieved by a previous call to the *mfe()* method, instructs all subsequent calls to scale the partition function accordingly.

**15.5.4.9   vrna_params_reset()**

```
void vrna_params_reset (
            vrna_fold_compound_t * vc,
            vrna_md_t * md_p )
```

`#include <ViennaRNA/params.h>`

Reset free energy parameters within a vrna_fold_compound_t according to provided, or default model details.

This function allows one to rescale free energy parameters for subsequent structure prediction or evaluation according to a set of model details, e.g. temperature values. To do so, the caller provides either a pointer to a set of model details to be used for rescaling, or NULL if global default setting should be used.

**See also**

vrna_exp_params_reset(), vrna_params_subs()

**Parameters**

| | |
|---|---|
| *vc* | The fold compound data structure |
| *md↩_p* | A pointer to the new model details (or NULL for reset to defaults) |

**SWIG Wrapper Notes** This function is attached to vrna_fc_s objects as overloaded **params_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

**15.5.4.10 vrna_exp_params_reset()**

```
vrna_exp_params_reset (
            vrna_fold_compound_t * vc,
            vrna_md_t * md_p )
```

`#include <ViennaRNA/params.h>`

Reset Boltzmann factors for partition function computations within a vrna_fold_compound_t according to provided, or default model details.

This function allows one to rescale Boltzmann factors for subsequent partition function computations according to a set of model details, e.g. temperature values. To do so, the caller provides either a pointer to a set of model details to be used for rescaling, or NULL if global default setting should be used.

**See also**

> vrna_params_reset(), vrna_exp_params_subst(), vrna_exp_params_rescale()

**Parameters**

| vc | The fold compound data structure |
|---|---|
| md↩ _p | A pointer to the new model details (or NULL for reset to defaults) |

**SWIG Wrapper Notes** This function is attached to vrna_fc_s objects as overloaded **exp_params_reset()** method.

> When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to vrna_exp_params_reset(), i.e. global default model settings are used. Passing an object of type vrna_md_s resets the fold compound according to the specifications stored within the vrna_md_s object.

**15.5.4.11 get_scaled_pf_parameters()**

```
vrna_exp_param_t* get_scaled_pf_parameters (
            void  )
```

`#include <ViennaRNA/params.h>`

get a data structure of type vrna_exp_param_t which contains the Boltzmann weights of several energy parameters scaled according to the current temperature

**Deprecated** Use vrna_exp_params() instead!

**Returns**

> The data structure containing Boltzmann weights for use in partition function calculations

**15.5.4.12 get_boltzmann_factors()**

```
vrna_exp_param_t* get_boltzmann_factors (
            double temperature,
            double betaScale,
            vrna_md_t md,
            double pf_scale )
```

```
#include <ViennaRNA/params.h>
```

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 $exp(-\Delta G/kT)$

**Deprecated** Use vrna_exp_params() instead!

**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

**15.5.4.13 get_boltzmann_factor_copy()**

```
vrna_exp_param_t* get_boltzmann_factor_copy (
            vrna_exp_param_t * parameters )
```

```
#include <ViennaRNA/params.h>
```

Get a copy of already precomputed Boltzmann factors.

**Deprecated** Use vrna_exp_params_copy() instead!

**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 data set

### 15.5.4.14 get_scaled_alipf_parameters()

vrna_exp_param_t* get_scaled_alipf_parameters (
            unsigned int *n_seq* )

#include <ViennaRNA/params.h>

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

**Deprecated** Use vrna_exp_params_comparative() instead!

### 15.5.4.15 get_boltzmann_factors_ali()

vrna_exp_param_t* get_boltzmann_factors_ali (
            unsigned int *n_seq,*
            double *temperature,*
            double *betaScale,*
            vrna_md_t *md,*
            double *pf_scale* )

#include <ViennaRNA/params.h>

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

**Deprecated** Use vrna_exp_params_comparative() instead!

**15.5.4.16 scale_parameters()**

vrna_param_t* scale_parameters (
           void  )

#include <ViennaRNA/params.h>

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.

**Deprecated** Use vrna_params() instead!

**Returns**

> A set of precomputed energy contributions

**15.5.4.17 get_scaled_parameters()**

vrna_param_t* get_scaled_parameters (
           double *temperature,*
           vrna_md_t *md* )

#include <ViennaRNA/params.h>

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

**Deprecated** Use vrna_params() instead!

**See also**

> vrna_md_t, set_model_details()

**Parameters**

| | |
|---|---|
| *temperature* | The temperature in degrees Celcius |
| *md* | The model details |

**Returns**

precomputed energy contributions and model settings

## 15.6 Extending the folding grammar with additional domains

This module covers simple and straight-forward extensions to the RNA folding grammar.

### 15.6.1 Detailed Description

This module covers simple and straight-forward extensions to the RNA folding grammar.

Collaboration diagram for Extending the folding grammar with additional domains:

**Modules**

- Unstructured domains

  *Add and modify unstructured domains to the RNA folding grammar.*
- Structured domains

  *Add and modify structured domains to the RNA folding grammar.*

## 15.7 Unstructured domains

Add and modify unstructured domains to the RNA folding grammar.

### 15.7.1 Detailed Description

Add and modify unstructured domains to the RNA folding grammar.

This module provides the tools to add and modify unstructured domains to the production rules of the RNA folding grammar. Usually this functionality is utilized for incorporating ligand binding to unpaired stretches of an RNA.

**Bug** Although the additional production rule(s) for unstructured domains as descibed in domains_unstructured are always treated as 'segments possibly bound to one or more ligands', the current implementation requires that at least one ligand is bound. The default implementation already takes care of the required changes, however, upon using callback functions other than the default ones, one has to take care of this fact. Please also note, that this behavior might change in one of the next releases, such that the decomposition schemes as shown above comply with the actual implementation.

A default implementation allows one to readily use this feature by simply adding sequence motifs and corresponding binding free energies with the function vrna_ud_add_motif() (see also Ligands binding to unstructured domains).

The grammar extension is realized using a callback function that

- evaluates the binding free energy of a ligand to its target sequence segment (white boxes in the figures above), or

- returns the free energy of an unpaired stretch possibly bound by a ligand, stored in the additional $U$ DP matrix.

The callback is passed the segment positions, the loop context, and which of the two above mentioned evaluations are required. A second callback implements the pre-processing step that prepares the $U$ DP matrix by evaluating all possible cases of the additional production rule. Both callbacks have a default implementation in *RNAlib*, but may be over-written by a user-implementation, making it fully user-customizable.

For equilibrium probability computations, two additional callbacks exist. One to store/add and one to retrieve the probability of unstructured domains at particular positions. Our implementation already takes care of computing the probabilities, but users of the unstructured domain feature are required to provide a mechanism to efficiently store/add the corresponding values into some external data structure. Collaboration diagram for Unstructured domains:

**Files**

- file unstructured_domains.h

  *Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.*

**Data Structures**

- struct vrna_unstructured_domain_s

  *Data structure to store all functionality for ligand binding. More...*

## Macros

- #define VRNA_UNSTRUCTURED_DOMAIN_EXT_LOOP 1U

    *Flag to indicate ligand bound to unpaired stretch in the exterior loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_HP_LOOP 2U

    *Flag to indicate ligand bound to unpaired stretch in a hairpin loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_INT_LOOP 4U

    *Flag to indicate ligand bound to unpaired stretch in an interior loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_MB_LOOP 8U

    *Flag to indicate ligand bound to unpaired stretch in a multibranch loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_MOTIF 16U

    *Flag to indicate ligand binding without additional unbound nucleotides (motif-only)*
- #define VRNA_UNSTRUCTURED_DOMAIN_ALL_LOOPS

    *Flag to indicate ligand bound to unpaired stretch in any loop (convenience macro)*

## Typedefs

- typedef struct vrna_unstructured_domain_s vrna_ud_t

    *Typename for the ligand binding extension data structure vrna_unstructured_domain_s.*
- typedef int() vrna_callback_ud_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)

    *Callback to retrieve binding free energy of a ligand bound to an unpaired sequence segment.*
- typedef FLT_OR_DBL() vrna_callback_ud_exp_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)

    *Callback to retrieve Boltzmann factor of the binding free energy of a ligand bound to an unpaired sequence segment.*
- typedef void() vrna_callback_ud_production(vrna_fold_compound_t *vc, void *data)

    *Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature.*
- typedef void() vrna_callback_ud_exp_production(vrna_fold_compound_t *vc, void *data)

    *Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature (partition function variant)*
- typedef void() vrna_callback_ud_probs_add(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, FLT_OR_DBL exp_energy, void *data)

    *Callback to store/add equilibrium probability for a ligand bound to an unpaired sequence segment.*
- typedef FLT_OR_DBL() vrna_callback_ud_probs_get(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, int motif, void *data)

    *Callback to retrieve equilibrium probability for a ligand bound to an unpaired sequence segment.*

## Functions

- void vrna_ud_add_motif (vrna_fold_compound_t *vc, const char *motif, double motif_en, unsigned int loop←↩_type)

    *Add an unstructured domain motif, e.g. for ligand binding.*
- void vrna_ud_remove (vrna_fold_compound_t *vc)

    *Remove ligand binding to unpaired stretches.*
- void vrna_ud_set_data (vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *free_cb)

    *Attach an auxiliary data structure.*
- void vrna_ud_set_prod_rule_cb (vrna_fold_compound_t *vc, vrna_callback_ud_production *pre_cb, vrna_callback_ud_energy *e_cb)

    *Attach production rule callbacks for free energies computations.*
- void vrna_ud_set_exp_prod_rule_cb (vrna_fold_compound_t *vc, vrna_callback_ud_exp_production *pre←↩_cb, vrna_callback_ud_exp_energy *exp_e_cb)

    *Attach production rule for partition function.*

## 15.7.2 Data Structure Documentation

### 15.7.2.1 struct vrna_unstructured_domain_s

Data structure to store all functionality for ligand binding.

**Data Fields**

- int uniq_motif_count

    *The unique number of motifs of different lengths.*
- unsigned int ∗ uniq_motif_size

    *An array storing a unique list of motif lengths.*
- int motif_count

    *Total number of distinguished motifs.*
- char ∗∗ motif

    *Motif sequences.*
- unsigned int ∗ motif_size

    *Motif lengths.*
- double ∗ motif_en

    *Ligand binding free energy contribution.*
- unsigned int ∗ motif_type

    *Type of motif, i.e. loop type the ligand binds to.*
- vrna_callback_ud_production ∗ prod_cb

    *Callback to ligand binding production rule, i.e. create/fill DP free energy matrices.*
- vrna_callback_ud_exp_production ∗ exp_prod_cb

    *Callback to ligand binding production rule, i.e. create/fill DP partition function matrices.*
- vrna_callback_ud_energy ∗ energy_cb

    *Callback to evaluate free energy of ligand binding to a particular unpaired stretch.*
- vrna_callback_ud_exp_energy ∗ exp_energy_cb

    *Callback to evaluate Boltzmann factor of ligand binding to a particular unpaired stretch.*
- void ∗ data

    *Auxiliary data structure passed to energy evaluation callbacks.*
- vrna_callback_free_auxdata ∗ free_data

    *Callback to free auxiliary data structure.*
- vrna_callback_ud_probs_add ∗ probs_add

    *Callback to store/add outside partition function.*
- vrna_callback_ud_probs_get ∗ probs_get

    *Callback to retrieve outside partition function.*

#### 15.7.2.1.1 Field Documentation

##### 15.7.2.1.1.1 prod_cb

vrna_callback_ud_production* vrna_unstructured_domain_s::prod_cb

Callback to ligand binding production rule, i.e. create/fill DP free energy matrices.

This callback will be executed right before the actual secondary structure decompositions, and, therefore, any implementation must not interleave with the regular DP matrices.

### 15.7.3 Typedef Documentation

#### 15.7.3.1 vrna_callback_ud_energy

```
typedef int() vrna_callback_ud_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int
loop_type, void *data)
```

`#include <ViennaRNA/unstructured_domains.h>`

Callback to retrieve binding free energy of a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions** This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand.

**Parameters**

| | |
|---|---|
| *vc* | The current vrna_fold_compound_t |
| *i* | The start of the unstructured domain (5' end) |
| *j* | The end of the unstructured domain (3' end) |
| *loop_type* | The loop context of the unstructured domain |
| *data* | Auxiliary data |

**Returns**

The auxiliary energy contribution in deka-cal/mol

#### 15.7.3.2 vrna_callback_ud_exp_energy

```
typedef FLT_OR_DBL() vrna_callback_ud_exp_energy(vrna_fold_compound_t *vc, int i, int j, unsigned
int loop_type, void *data)
```

`#include <ViennaRNA/unstructured_domains.h>`

Callback to retrieve Boltzmann factor of the binding free energy of a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions** This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand (Partition function variant, i.e. the Boltzmann factors instead of actual free energies).

**Parameters**

| | |
|---|---|
| *vc* | The current vrna_fold_compound_t |
| *i* | The start of the unstructured domain (5' end) |
| *j* | The end of the unstructured domain (3' end) |
| *loop_type* | The loop context of the unstructured domain |
| *data* | Auxiliary data |

**Returns**

The auxiliary energy contribution as Boltzmann factor

### 15.7.3.3   vrna_callback_ud_production

typedef void() vrna_callback_ud_production([vrna_fold_compound_t](#) *vc, void *data)

#include <[ViennaRNA/unstructured_domains.h](#)>

Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature.

**Notes on Callback Functions**  The production rule for the unstructured domain grammar extension

### 15.7.3.4   vrna_callback_ud_exp_production

typedef void() vrna_callback_ud_exp_production([vrna_fold_compound_t](#) *vc, void *data)

#include <[ViennaRNA/unstructured_domains.h](#)>

Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature (partition function variant)

**Notes on Callback Functions**  The production rule for the unstructured domain grammar extension (Partition function variant)

### 15.7.3.5   vrna_callback_ud_probs_add

typedef void() vrna_callback_ud_probs_add([vrna_fold_compound_t](#) *vc, int i, int j, unsigned int loop_type, [FLT_OR_DBL](#) exp_energy, void *data)

#include <[ViennaRNA/unstructured_domains.h](#)>

Callback to store/add equilibrium probability for a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions**  A callback function to store equilibrium probabilities for the unstructured domain feature

### 15.7.3.6   vrna_callback_ud_probs_get

```
typedef FLT_OR_DBL() vrna_callback_ud_probs_get(vrna_fold_compound_t *vc, int i, int j, unsigned
int loop_type, int motif, void *data)
```

`#include <ViennaRNA/unstructured_domains.h>`

Callback to retrieve equilibrium probability for a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions**   A callback function to retrieve equilibrium probabilities for the unstructured domain feature

## 15.7.4   Function Documentation

### 15.7.4.1   vrna_ud_add_motif()

```
void vrna_ud_add_motif (
            vrna_fold_compound_t * vc,
            const char * motif,
            double motif_en,
            unsigned int loop_type )
```

`#include <ViennaRNA/unstructured_domains.h>`

Add an unstructured domain motif, e.g. for ligand binding.

This function adds a ligand binding motif and the associated binding free energy to the vrna_ud_t attribute of a vrna_fold_compound_t. The motif data will then be used in subsequent secondary structure predictions. Multiple calls to this function with different motifs append all additional data to a list of ligands, which all will be evaluated. Ligand motif data can be removed from the vrna_fold_compound_t again using the vrna_ud_remove() function. The loop type parameter allows one to limit the ligand binding to particular loop type, such as the exterior loop, hairpin loops, interior loops, or multibranch loops.

**See also**

> VRNA_UNSTRUCTURED_DOMAIN_EXT_LOOP, VRNA_UNSTRUCTURED_DOMAIN_HP_LOOP, VRNA_UNSTRUCTURED
> VRNA_UNSTRUCTURED_DOMAIN_MB_LOOP,        VRNA_UNSTRUCTURED_DOMAIN_ALL_LOOPS,
> vrna_ud_remove()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t data structure the ligand motif should be bound to |
| *motif* | The sequence motif the ligand binds to |
| *motif_en* | The binding free energy of the ligand in kcal/mol |
| *loop_type* | The loop type the ligand binds to |

**SWIG Wrapper Notes**   This function is attached as method **ud_add_motif()** to objects of type *fold_compound*

**15.7.4.2 vrna_ud_remove()**

```
void vrna_ud_remove (
            vrna_fold_compound_t * vc )
```

`#include <ViennaRNA/unstructured_domains.h>`

Remove ligand binding to unpaired stretches.

This function removes all ligand motifs that were bound to a vrna_fold_compound_t using the vrna_ud_add_motif() function.

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t data structure the ligand motif data should be removed from |

**SWIG Wrapper Notes** This function is attached as method **ud_remove()** to objects of type *fold_compound*

**15.7.4.3 vrna_ud_set_data()**

```
void vrna_ud_set_data (
            vrna_fold_compound_t * vc,
            void * data,
            vrna_callback_free_auxdata * free )
```

`#include <ViennaRNA/unstructured_domains.h>`

Attach an auxiliary data structure.

This function binds an arbitrary, auxiliary data structure for user-implemented ligand binding. The optional callback `free` will be passed the bound data structure whenever the vrna_fold_compound_t is removed from memory to avoid memory leaks.

**See also**

vrna_ud_set_prod_rule_cb(), vrna_ud_set_exp_prod_rule_cb(), vrna_ud_remove()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t data structure the auxiliary data structure should be bound to |
| *data* | A pointer to the auxiliary data structure |
| *free_cb* | A pointer to a callback function that free's memory occupied by `data` |

**SWIG Wrapper Notes** This function is attached as method **ud_set_data()** to objects of type *fold_compound*

### 15.7.4.4 vrna_ud_set_prod_rule_cb()
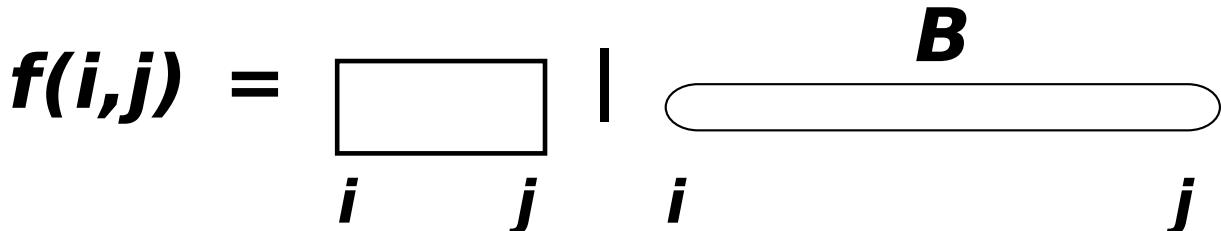
```
void vrna_ud_set_prod_rule_cb (
            vrna_fold_compound_t * vc,
            vrna_callback_ud_production * pre_cb,
            vrna_callback_ud_energy * e_cb )
```

`#include <ViennaRNA/unstructured_domains.h>`
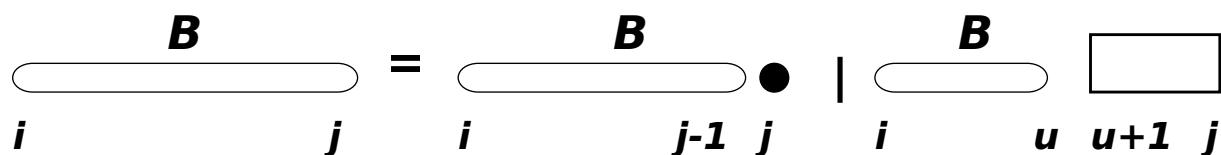
Attach production rule callbacks for free energies computations.

Use this function to bind a user-implemented grammar extension for unstructured domains.

The callback `e_cb` needs to evaluate the free energy contribution $f(i,j)$ of the unpaired segment $[i,j]$. It will be executed in each of the regular secondary structure production rules. Whenever the callback is passed the VRNA_UNSTRUCTURED_DOMAIN_MOTIF flag via its `loop_type` parameter the contribution of any ligand that consecutively binds from position $i$ to $j$ (the white box) is requested. Otherwise, the callback usually performs a lookup in the precomputed B matrices. Which B matrix is addressed will be indicated by the flags VRNA_UNSTRUCTURED_DOMAIN_EXT_LOOP, VRNA_UNSTRUCTURED_DOMAIN_HP_LOOP VRNA_UNSTRUCTURED_DOMAIN_INT_LOOP, and VRNA_UNSTRUCTURED_DOMAIN_MB_LOOP. As their names already imply, they specify exterior loops (F production rule), hairpin loops and interior loops (C production rule), and multibranch loops (M and M1 production rule).



The `pre_cb` callback will be executed as a pre-processing step right before the regular secondary structure rules. Usually one would use this callback to fill the dynamic programming matrices U and preparations of the auxiliary data structure vrna_unstructured_domain_s.data



**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t data structure the callback will be bound to |
| *pre_cb* | A pointer to a callback function for the B production rule |
| *e_cb* | A pointer to a callback function for free energy evaluation |

**15.7.4.5 vrna_ud_set_exp_prod_rule_cb()**

```
void vrna_ud_set_exp_prod_rule_cb (
            vrna_fold_compound_t * vc,
            vrna_callback_ud_exp_production * pre_cb,
            vrna_callback_ud_exp_energy * exp_e_cb )
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Attach production rule for partition function.

This function is the partition function companion of vrna_ud_set_prod_rule_cb().

Use it to bind callbacks to (i) fill the U production rule dynamic programming matrices and/or prepare the vrna_unstructured_domain_s.data, and (ii) provide a callback to retrieve partition functions for subsegments $[i, j]$.



**See also**

vrna_ud_set_prod_rule_cb()

**Parameters**

| vc | The vrna_fold_compound_t data structure the callback will be bound to |
|---|---|
| pre_cb | A pointer to a callback function for the B production rule |
| exp_e_cb | A pointer to a callback function that retrieves the partition function for a segment $[i, j]$ that may be bound by one or more ligands. |

**SWIG Wrapper Notes** This function is attached as method **ud_set_exp_prod_rule_cb()** to objects of type *fold↩ _compound*

## 15.8 Structured domains

Add and modify structured domains to the RNA folding grammar.

### 15.8.1 Detailed Description

Add and modify structured domains to the RNA folding grammar.

This module provides the tools to add and modify structured domains to the production rules of the RNA folding grammar. Usually this functionality is utilized for incorporating self-enclosed structural modules that exhibit a more or less complex base pairing pattern. Collaboration diagram for Structured domains:

**Files**

- file structured_domains.h

  *This module provides interfaces that deal with additional structured domains in the folding grammar.*

## 15.9 Constraining the RNA folding grammar

This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.

### 15.9.1 Detailed Description
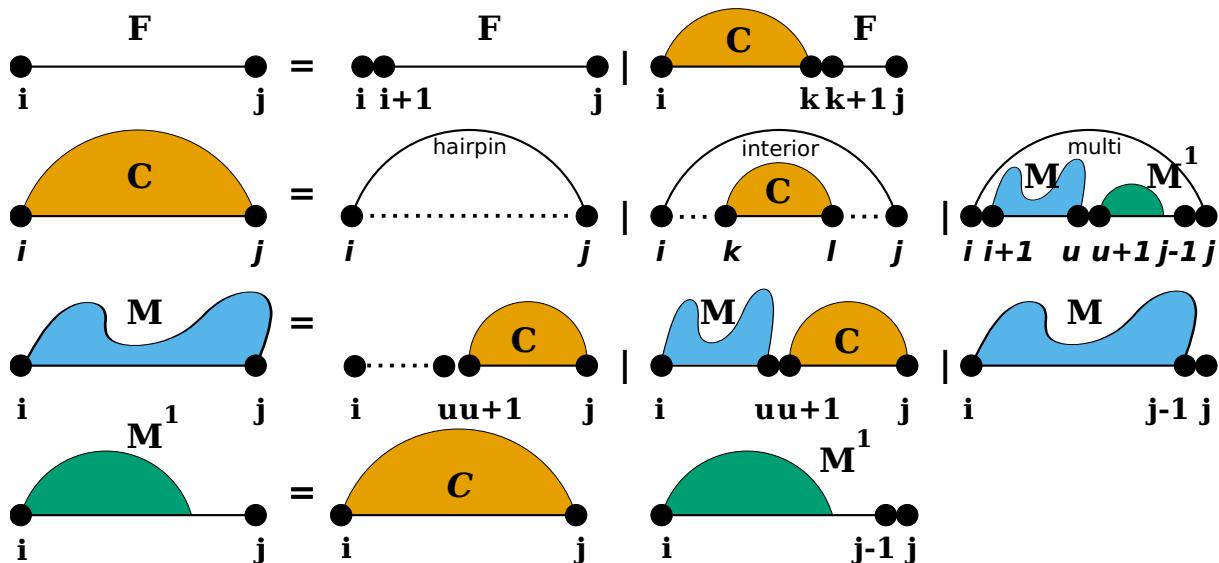
This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.

Secondary Structure constraints can be subdivided into two groups:

- Hard constraints, and

- Soft constraints.

While Hard-Constraints directly influence the production rules used in the folding recursions by allowing, disallowing, or enforcing certain decomposition steps, Soft-constraints on the other hand are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations.

Secondary structure constraints are always applied at decomposition level, i.e. in each step of the recursive structure decomposition, for instance during MFE prediction. Below is a visualization of the decomposition scheme



For Hard constraints the following option flags may be used to constrain the pairing behavior of single, or pairs of nucleotides:

- VRNA_CONSTRAINT_CONTEXT_EXT_LOOP - Hard constraints flag, base pair in the exterior loop.

- VRNA_CONSTRAINT_CONTEXT_HP_LOOP - Hard constraints flag, base pair encloses hairpin loop.

- VRNA_CONSTRAINT_CONTEXT_INT_LOOP - Hard constraints flag, base pair encloses an interior loop.

- VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC - Hard constraints flag, base pair encloses a multi branch loop.

- VRNA_CONSTRAINT_CONTEXT_MB_LOOP - Hard constraints flag, base pair is enclosed in an interior loop.

- VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC - Hard constraints flag, base pair is enclosed in a multi branch loop.

- VRNA_CONSTRAINT_CONTEXT_ENFORCE - Hard constraint flag to indicate enforcement of constraints.

- VRNA_CONSTRAINT_CONTEXT_NO_REMOVE - Hard constraint flag to indicate not to remove base pairs that conflict with a given constraint.

- VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS - Constraint context flag indicating any loop context.

However, for Soft constraints we do not allow for simple loop type dependent constraining. But soft constraints are equipped with generic constraint support. This enables the user to pass arbitrary callback functions that return auxiliary energy contributions for evaluation the evaluation of any decomposition.

The callback will then always be notified about the type of decomposition that is happening, and the corresponding delimiting sequence positions. The following decomposition steps are distinguished, and should be captured by the user's implementation of the callback:

- VRNA_DECOMP_PAIR_HP - Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.

- VRNA_DECOMP_PAIR_IL - Indicator for interior loop decomposition step.

- VRNA_DECOMP_PAIR_ML - Indicator for multibranch loop decomposition step.

- VRNA_DECOMP_ML_ML_ML - Indicator for decomposition of multibranch loop part.

- VRNA_DECOMP_ML_STEM - Indicator for decomposition of multibranch loop part.

- VRNA_DECOMP_ML_ML - Indicator for decomposition of multibranch loop part.

- VRNA_DECOMP_ML_UP - Indicator for decomposition of multibranch loop part.

- VRNA_DECOMP_ML_ML_STEM - Indicator for decomposition of multibranch loop part.

- VRNA_DECOMP_ML_COAXIAL - Indicator for decomposition of multibranch loop part.

- VRNA_DECOMP_EXT_EXT - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_UP - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_STEM - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_EXT_EXT - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_STEM_EXT - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_STEM_OUTSIDE - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_EXT_STEM - Indicator for decomposition of exterior loop part.

- VRNA_DECOMP_EXT_EXT_STEM1 - Indicator for decomposition of exterior loop part.

Simplified interfaces to the soft constraints framework can be obtained by the implementations in the submodules

- SHAPE reactivity data and

- ligands.

An implementation that generates soft constraints for unpaired nucleotides by minimizing the discrepancy between their predicted and expected pairing probability is available in submodule Generate soft constraints from data. Collaboration diagram for Constraining the RNA folding grammar:

**Modules**

- Hard constraints

    *This module covers all functionality for hard constraints in secondary structure prediction.*
- Soft constraints

    *Functions and data structures for secondary structure soft constraints.*

**Files**

- file constraints.h

    *Functions and data structures for constraining secondary structure predictions and evaluation.*

**Macros**

- #define VRNA_CONSTRAINT_FILE 0

    *Flag for vrna_constraints_add() to indicate that constraints are present in a text file.*
- #define VRNA_CONSTRAINT_SOFT_MFE 0

    *Indicate generation of constraints for MFE folding.*
- #define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF

    *Indicate generation of constraints for partition function computation.*
- #define VRNA_DECOMP_PAIR_HP (unsigned char)1

    *Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.*
- #define VRNA_DECOMP_PAIR_IL (unsigned char)2

    *Indicator for interior loop decomposition step.*
- #define VRNA_DECOMP_PAIR_ML (unsigned char)3

    *Indicator for multibranch loop decomposition step.*
- #define VRNA_DECOMP_ML_ML_ML (unsigned char)5

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_STEM (unsigned char)6

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_ML (unsigned char)7

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_UP (unsigned char)8

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_ML_STEM (unsigned char)9

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_COAXIAL (unsigned char)10

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_COAXIAL_ENC (unsigned char)11

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_EXT_EXT (unsigned char)12

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_UP (unsigned char)13

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_STEM (unsigned char)14

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_EXT_EXT (unsigned char)15

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_STEM_EXT (unsigned char)16

*Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_STEM_OUTSIDE (unsigned char)17

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_EXT_STEM (unsigned char)18

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_EXT_STEM1 (unsigned char)19

    *Indicator for decomposition of exterior loop part.*

**Functions**

- void vrna_constraints_add (vrna_fold_compound_t ∗vc, const char ∗constraint, unsigned int options)

    *Add constraints to a vrna_fold_compound_t data structure.*
- void vrna_message_constraint_options (unsigned int option)

    *Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)*
- void vrna_message_constraint_options_all (void)

    *Print structure constraint characters to stdout (full constraint support)*

## 15.9.2 Macro Definition Documentation

### 15.9.2.1 VRNA_CONSTRAINT_FILE

```
#define VRNA_CONSTRAINT_FILE 0
```

```
#include <ViennaRNA/constraints.h>
```

Flag for vrna_constraints_add() to indicate that constraints are present in a text file.

**See also**

> vrna_constraints_add()

**Deprecated** Use 0 instead!

### 15.9.2.2 VRNA_CONSTRAINT_SOFT_MFE

```
#define VRNA_CONSTRAINT_SOFT_MFE 0
```

```
#include <ViennaRNA/constraints.h>
```

Indicate generation of constraints for MFE folding.

**Deprecated** This flag has no meaning anymore, since constraints are now always stored!

### 15.9.2.3 VRNA_CONSTRAINT_SOFT_PF

#define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF

#include <ViennaRNA/constraints.h>

Indicate generation of constraints for partition function computation.

**Deprecated** Use VRNA_OPTION_PF instead!
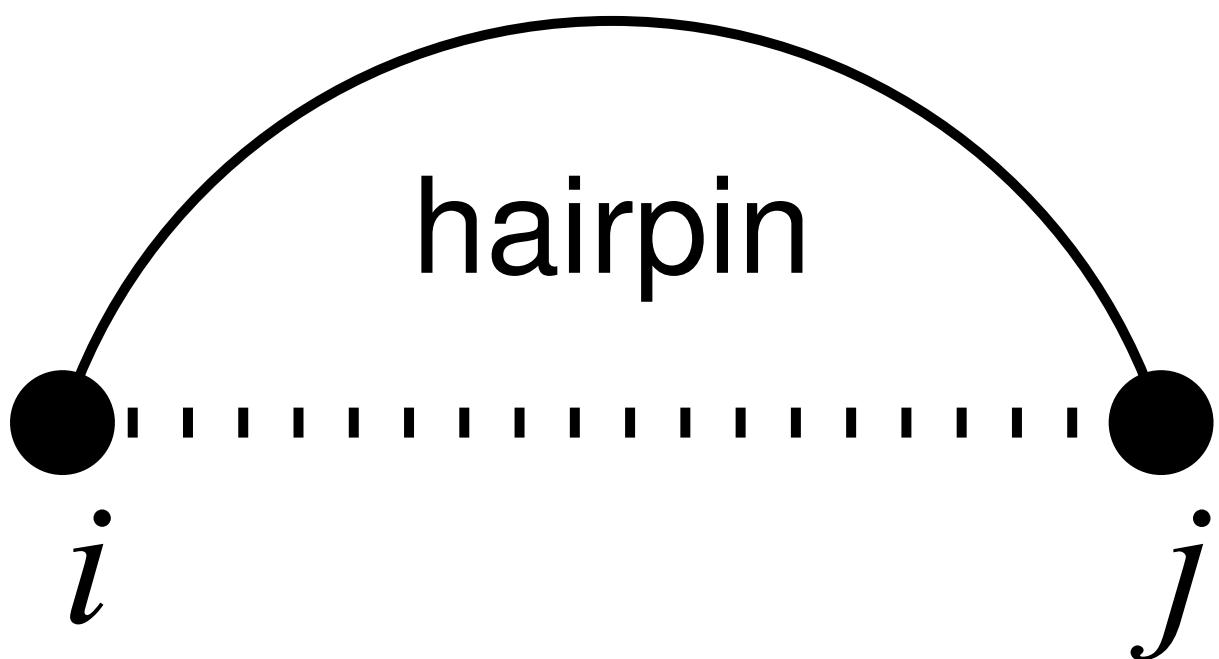
### 15.9.2.4 VRNA_DECOMP_PAIR_HP

#define VRNA_DECOMP_PAIR_HP (unsigned char)1

#include <ViennaRNA/constraints.h>

Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a hairpin loop enclosed by the base pair $(i, j)$.
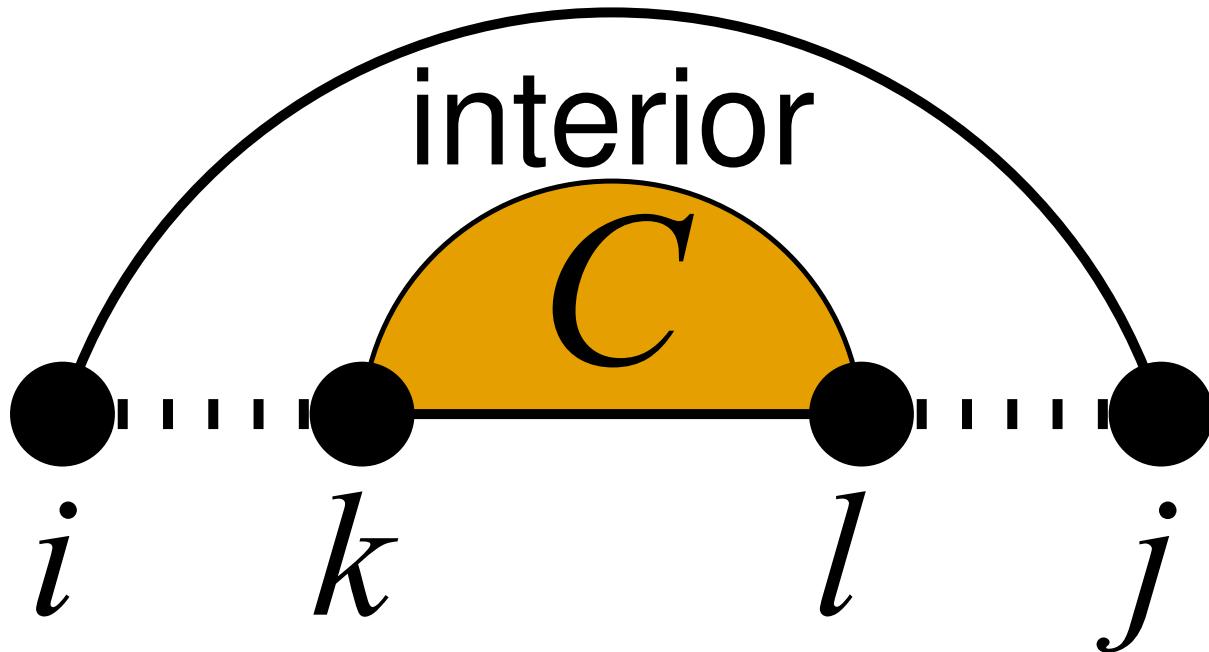
**15.9.2.5 VRNA_DECOMP_PAIR_IL**

```
#define VRNA_DECOMP_PAIR_IL (unsigned char)2
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for interior loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an interior loop enclosed by the base pair $(i, j)$, and enclosing the base pair $(k, l)$.
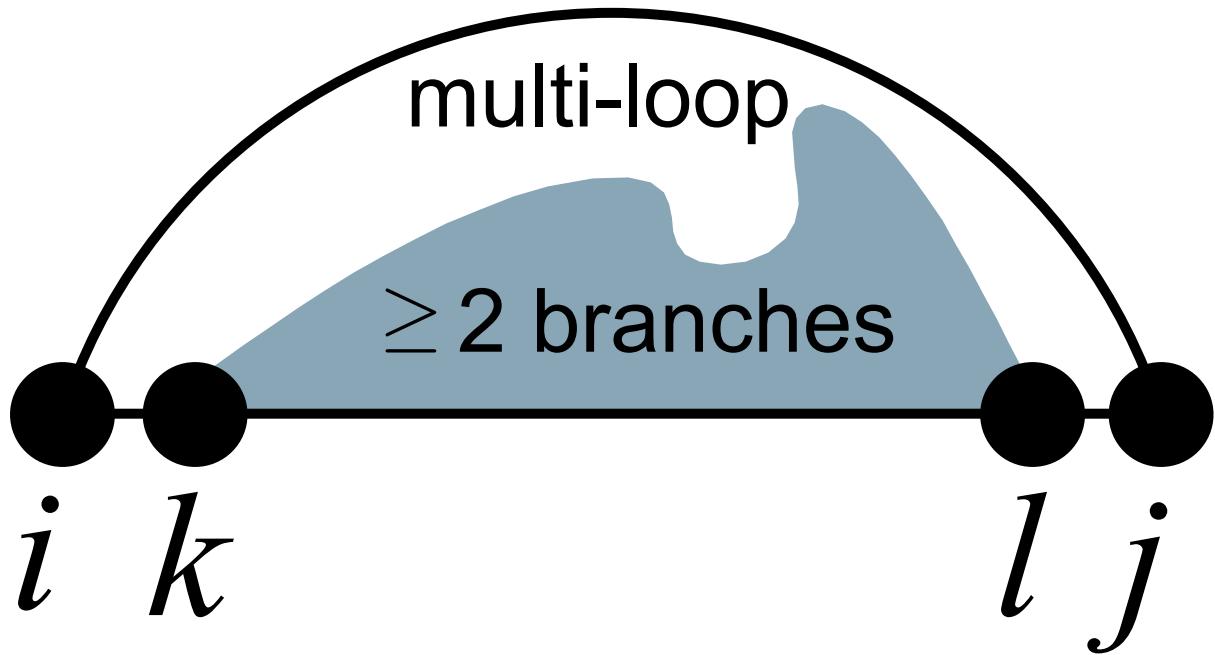


**15.9.2.6 VRNA_DECOMP_PAIR_ML**

```
#define VRNA_DECOMP_PAIR_ML (unsigned char)3
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for multibranch loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multi-branch loop enclosed by the base pair $(i, j)$, and consisting of some enclosed multi loop content from k to l.
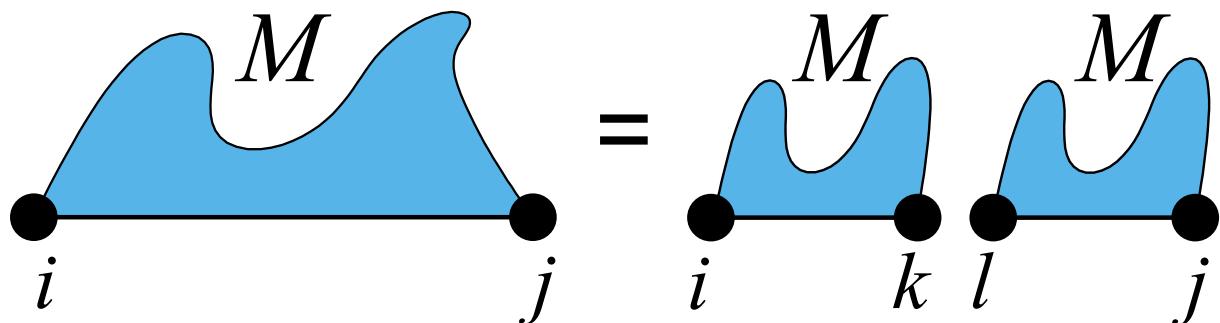
### 15.9.2.7 VRNA_DECOMP_ML_ML_ML

```
#define VRNA_DECOMP_ML_ML_ML (unsigned char)5
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multi-branch loop part in the interval $[i : j]$, which will be decomposed into two multibranch loop parts $[i : k]$, and $[l : j]$.
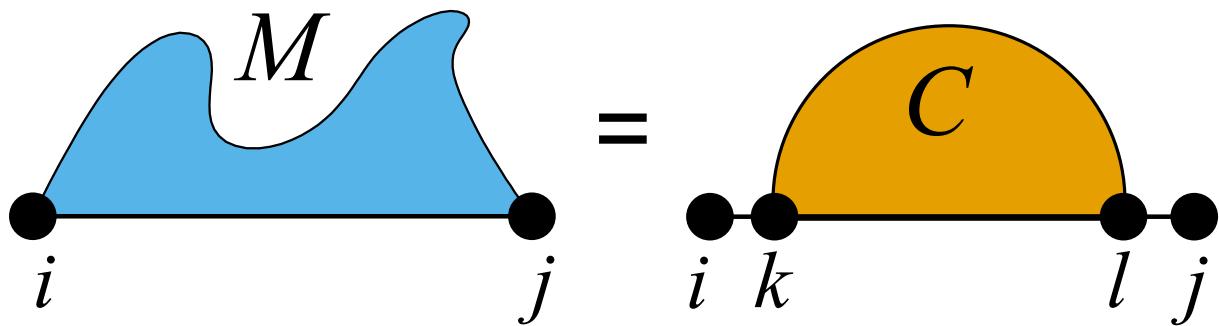
### 15.9.2.8 VRNA_DECOMP_ML_STEM

`#define VRNA_DECOMP_ML_STEM (unsigned char)6`

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i:j]$, which will be considered a single stem branching off with base pair $(k, l)$.
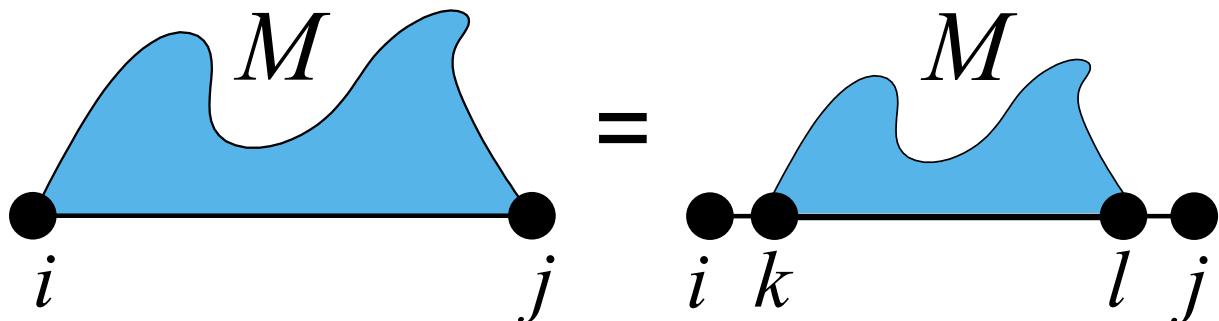


### 15.9.2.9 VRNA_DECOMP_ML_ML

`#define VRNA_DECOMP_ML_ML (unsigned char)7`

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i:j]$, which will be decomposed into a (usually) smaller multibranch loop part $[k:l]$.
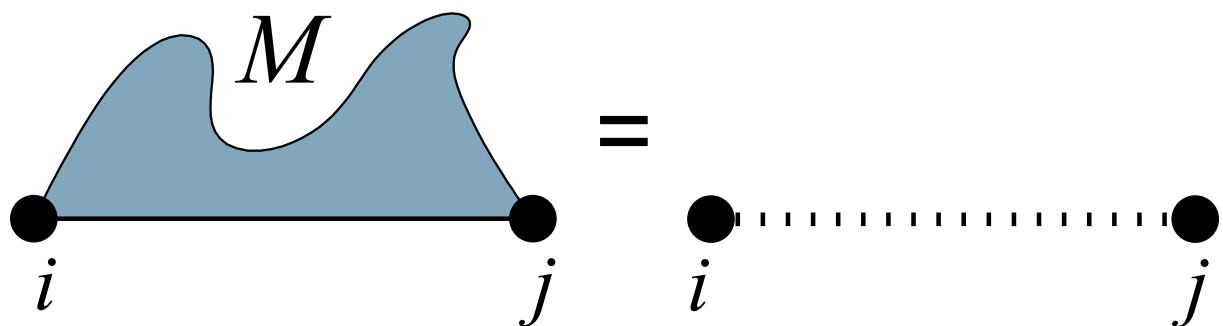
### 15.9.2.10 VRNA_DECOMP_ML_UP

`#define VRNA_DECOMP_ML_UP (unsigned char)8`

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multi-branch loop part in the interval $[i : j]$, which will be considered a multibranch loop part that only consists of unpaired nucleotides.
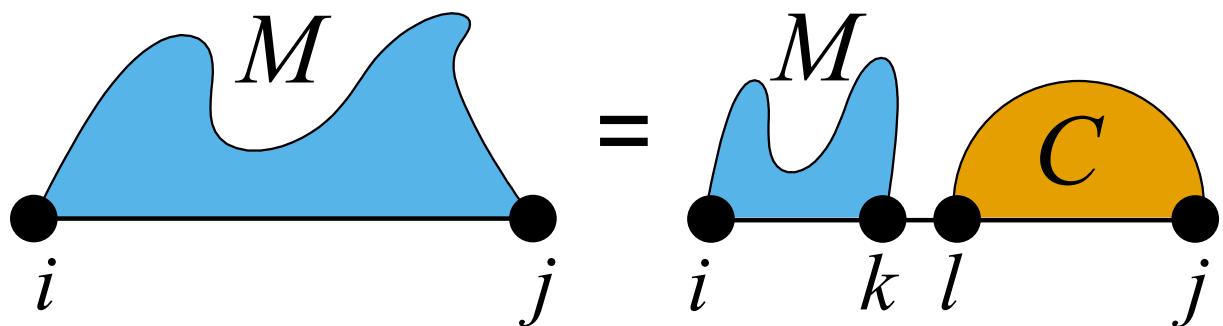


### 15.9.2.11 VRNA_DECOMP_ML_ML_STEM

`#define VRNA_DECOMP_ML_ML_STEM (unsigned char)9`

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multi-branch loop part in the interval $[i : j]$, which will decomposed into a multibranch loop part $[i : k]$, and a stem with enclosing base pair $(l, j)$.
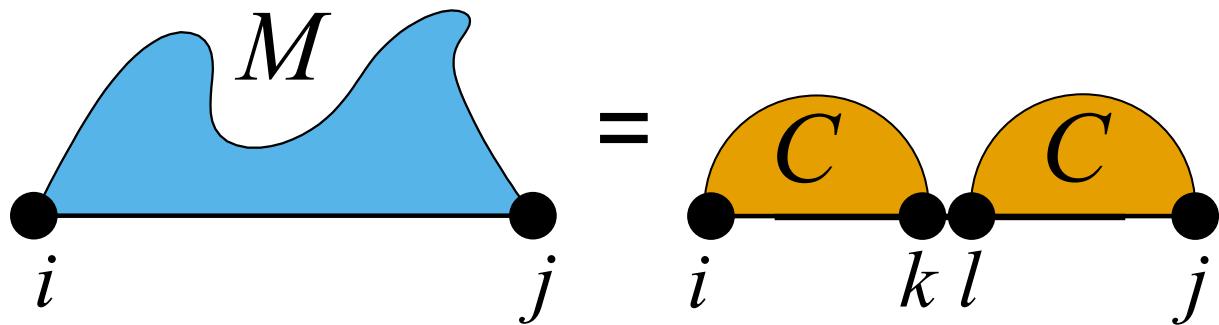
### 15.9.2.12 VRNA_DECOMP_ML_COAXIAL

```
#define VRNA_DECOMP_ML_COAXIAL (unsigned char)10
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, where two stems with enclosing pairs $(i, k)$ and $(l, j)$ are coaxially stacking onto each other.
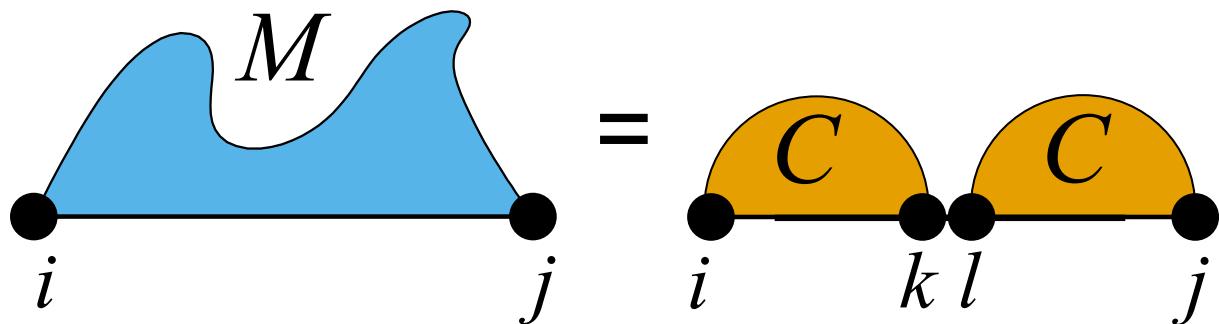


### 15.9.2.13 VRNA_DECOMP_ML_COAXIAL_ENC

```
#define VRNA_DECOMP_ML_COAXIAL_ENC (unsigned char)11
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval $[i : j]$, where two stems with enclosing pairs $(i, k)$ and $(l, j)$ are coaxially stacking onto each other.
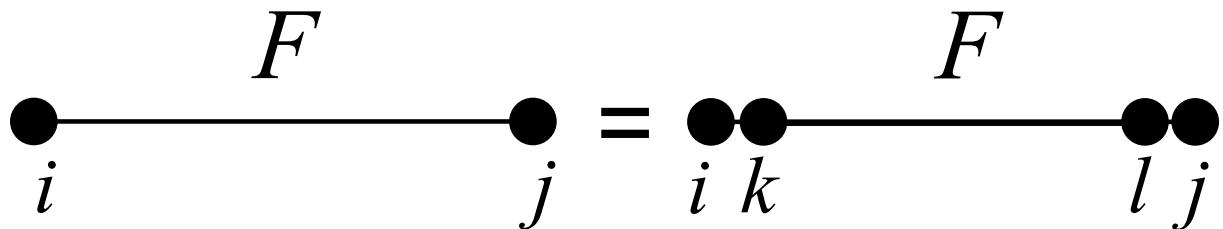
**15.9.2.14 VRNA_DECOMP_EXT_EXT**

`#define VRNA_DECOMP_EXT_EXT (unsigned char)12`

`#include <`ViennaRNA/constraints.h`>`

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into a (usually) smaller exterior loop part $[k : l]$.
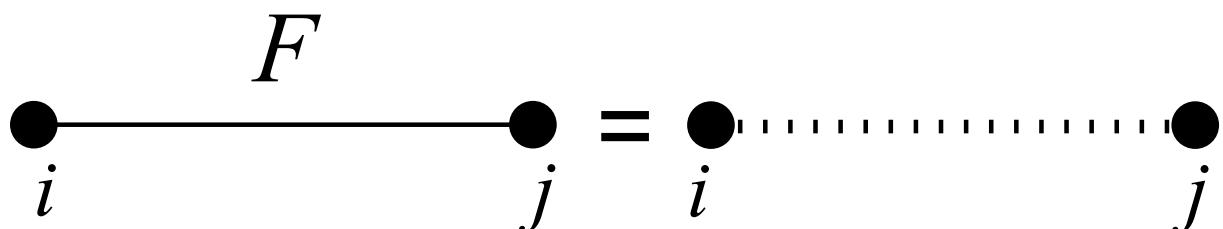


**15.9.2.15 VRNA_DECOMP_EXT_UP**

`#define VRNA_DECOMP_EXT_UP (unsigned char)13`

`#include <`ViennaRNA/constraints.h`>`

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be considered as an exterior loop component consisting of only unpaired nucleotides.
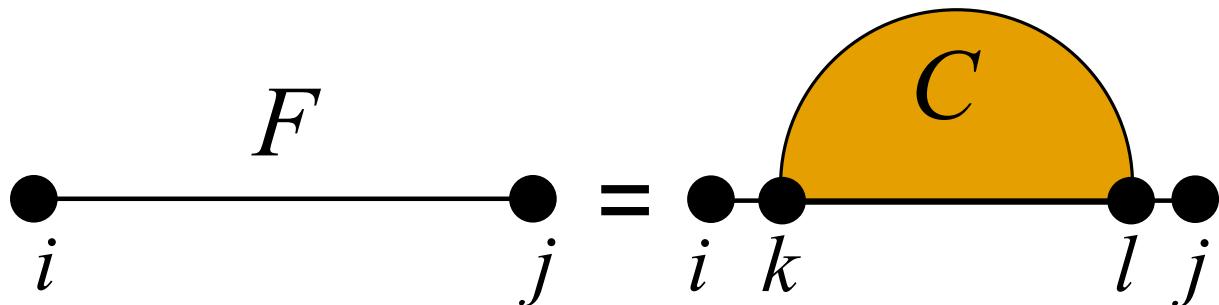
### 15.9.2.16 VRNA_DECOMP_EXT_STEM

```
#define VRNA_DECOMP_EXT_STEM (unsigned char)14
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be considered a stem with enclosing pair $(k, l)$.
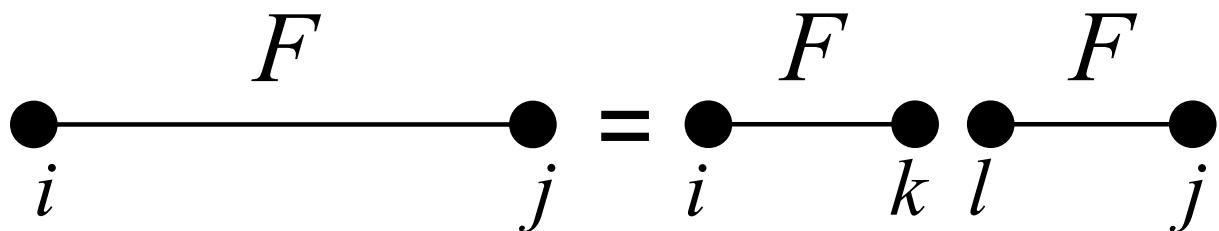


### 15.9.2.17 VRNA_DECOMP_EXT_EXT_EXT

```
#define VRNA_DECOMP_EXT_EXT_EXT (unsigned char)15
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into two exterior loop parts $[i : k]$ and $[l : j]$.
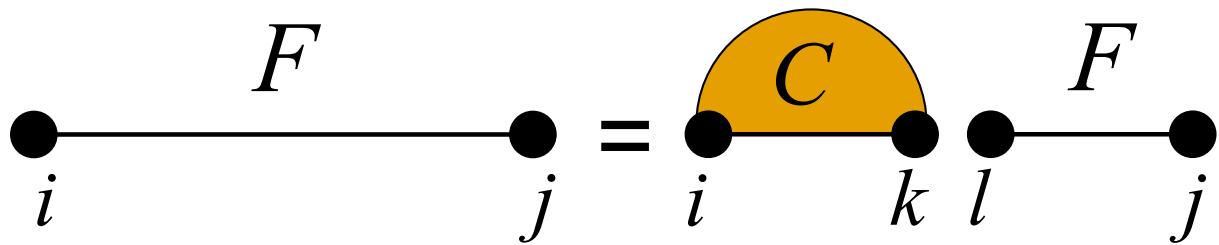
### 15.9.2.18 VRNA_DECOMP_EXT_STEM_EXT

```
#define VRNA_DECOMP_EXT_STEM_EXT (unsigned char)16
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into a stem branching off with base pair $(i, k)$, and an exterior loop part $[l : j]$.
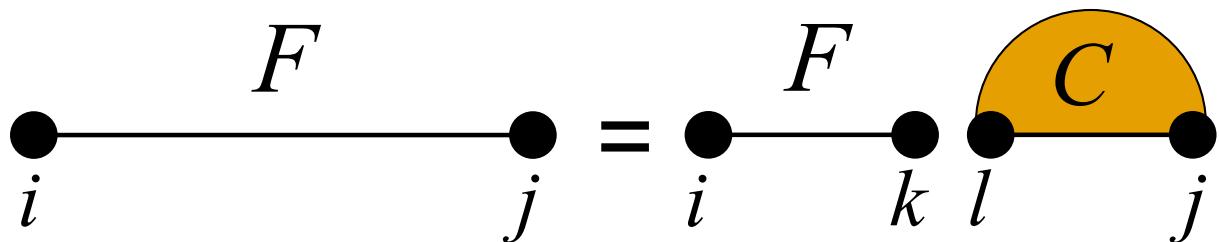


### 15.9.2.19 VRNA_DECOMP_EXT_EXT_STEM

```
#define VRNA_DECOMP_EXT_EXT_STEM (unsigned char)18
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into an exterior loop part $[i : k]$, and a stem branching off with base pair $(l, j)$.
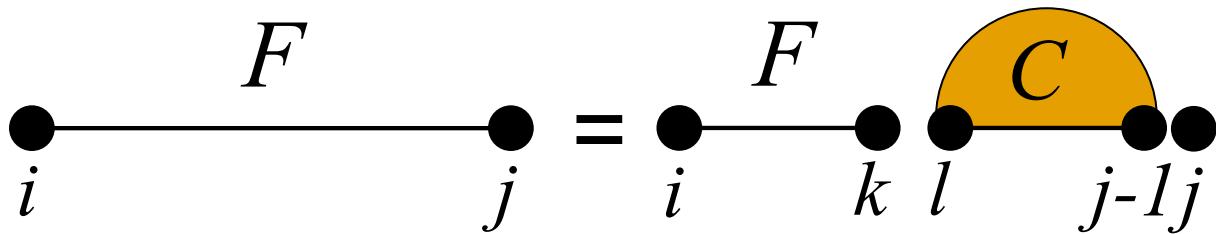
#### 15.9.2.20 VRNA_DECOMP_EXT_EXT_STEM1

```
#define VRNA_DECOMP_EXT_EXT_STEM1 (unsigned char)19
```

```
#include <ViennaRNA/constraints.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval $[i : j]$, which will be decomposed into an exterior loop part $[i : k]$, and a stem branching off with base pair $(l, j - 1)$.



### 15.9.3 Function Documentation

#### 15.9.3.1 vrna_constraints_add()

```
void vrna_constraints_add (
            vrna_fold_compound_t * vc,
            const char * constraint,
            unsigned int options )
```

```
#include <ViennaRNA/constraints.h>
```

Add constraints to a vrna_fold_compound_t data structure.

Use this function to add/update the hard/soft constraints The function allows for passing a string 'constraint' that can either be a filename that points to a constraints definition file or it may be a pseudo dot-bracket notation indicating hard constraints. For the latter, the user has to pass the VRNA_CONSTRAINT_DB option. Also, the user has to specify, which characters are allowed to be interpreted as constraints by passing the corresponding options via the third parameter.

**See also**

> vrna_hc_init(), vrna_hc_add_up(), vrna_hc_add_up_batch(), vrna_hc_add_bp(), vrna_sc_init(), vrna_sc_set_up(), vrna_sc_set_bp(), vrna_sc_add_SHAPE_deigan(), vrna_sc_add_SHAPE_zarringhalam(), vrna_hc_free(), vrna_sc_free(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_DEFAULT, VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTRAMOL, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_GQUAD

The following is an example for adding hard constraints given in pseudo dot-bracket notation. Here, `vc` is the vrna_fold_compound_t object, `structure` is a char array with the hard constraint in dot-bracket notation, and `enforceConstraints` is a flag indicating whether or not constraints for base pairs should be enforced instead of just doing a removal of base pair that conflict with the constraint.

```
    unsigned int constraint_options = VRNA_CONSTRAINT_DB_DEFAULT;

    if (enforceConstraints)
      constraint_options |= VRNA_CONSTRAINT_DB_ENFORCE_BP;

    if (canonicalBPonly)
      constraint_options |= VRNA_CONSTRAINT_DB_CANONICAL_BP;

    vrna_constraints_add(fc, (const char *)cstruc, constraint_options);
```

In constrat to the above, constraints may also be read from file:

```
    vrna_constraints_add(fc, constraints_file,
      VRNA_OPTION_DEFAULT);
```

**See also**

> vrna_hc_add_from_db(), vrna_hc_add_up(), vrna_hc_add_up_batch() vrna_hc_add_bp_unspecific(),
> vrna_hc_add_bp()

**Parameters**

| vc | The fold compound |
|---|---|
| constraint | A string with either the filename of the constraint definitions or a pseudo dot-bracket notation of the hard constraint. May be NULL. |
| options | The option flags |

**15.9.3.2 vrna_message_constraint_options()**

```
void vrna_message_constraint_options (
            unsigned int option )
```

`#include <`ViennaRNA/constraints_hard.h`>`

Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)

Currently available options are:
VRNA_CONSTRAINT_DB_PIPE (paired with another base)
VRNA_CONSTRAINT_DB_DOT (no constraint at all)
VRNA_CONSTRAINT_DB_X (base must not pair)
VRNA_CONSTRAINT_DB_ANG_BRACK (paired downstream/upstream)
VRNA_CONSTRAINT_DB_RND_BRACK (base i pairs base j)
pass a collection of options as one value like this:

```
vrna_message_constraints(option_1 | option_2 | option_n)
```

**See also**

> vrna_message_constraint_options_all(), vrna_constraints_add(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_PIPE,
> VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK,
> VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_INTRAMOL

**Parameters**

| | |
|---|---|
| *option* | Option switch that tells which constraint help will be printed |

### 15.9.3.3 vrna_message_constraint_options_all()

```
void vrna_message_constraint_options_all (
            void  )
```

#include <ViennaRNA/constraints_hard.h>

Print structure constraint characters to stdout (full constraint support)

**See also**

vrna_message_constraint_options(), vrna_constraints_add(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTERMOL, VRNA_CONSTRAINT_DB_INTRAMOL

## 15.10 Hard constraints

This module covers all functionality for hard constraints in secondary structure prediction.

### 15.10.1 Detailed Description

This module covers all functionality for hard constraints in secondary structure prediction.

Collaboration diagram for Hard constraints:

**Files**

- file constraints_hard.h

    *Functions and data structures for handling of secondary structure hard constraints.*

**Data Structures**

- struct vrna_hc_s

    *The hard constraints data structure. More...*
- struct vrna_hc_up_s

    *A single hard constraint for a single nucleotide. More...*

**Macros**

- #define VRNA_CONSTRAINT_DB 16384U

    *Flag for vrna_constraints_add() to indicate that constraint is passed in pseudo dot-bracket notation.*
- #define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U

    *Switch for dot-bracket structure constraint to enforce base pairs.*
- #define VRNA_CONSTRAINT_DB_PIPE 65536U

    *Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.*
- #define VRNA_CONSTRAINT_DB_DOT 131072U

    *dot '.' switch for structure constraints (no constraint at all)*
- #define VRNA_CONSTRAINT_DB_X 262144U

    *'x' switch for structure constraint (base must not pair)*
- #define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U

    *round brackets '(',')' switch for structure constraint (base i pairs base j)*
- #define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U

    *Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.*
- #define VRNA_CONSTRAINT_DB_INTERMOL 4194304U

    *Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.*
- #define VRNA_CONSTRAINT_DB_GQUAD 8388608U

    *'+' switch for structure constraint (base is involved in a gquad)*
- #define VRNA_CONSTRAINT_DB_WUSS 33554432U

    *Flag to indicate Washington University Secondary Structure (WUSS) notation of the hard constraint string.*
- #define VRNA_CONSTRAINT_DB_DEFAULT

*Switch for dot-bracket structure constraint with default symbols.*

- #define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP (unsigned char)0x01

    *Hard constraints flag, base pair in the exterior loop.*

- #define VRNA_CONSTRAINT_CONTEXT_HP_LOOP (unsigned char)0x02

    *Hard constraints flag, base pair encloses hairpin loop.*

- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP (unsigned char)0x04

    *Hard constraints flag, base pair encloses an interior loop.*

- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC (unsigned char)0x08

    *Hard constraints flag, base pair encloses a multi branch loop.*

- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP (unsigned char)0x10

    *Hard constraints flag, base pair is enclosed in an interior loop.*

- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC (unsigned char)0x20

    *Hard constraints flag, base pair is enclosed in a multi branch loop.*

- #define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

    *Constraint context flag indicating any loop context.*

## Typedefs

- typedef struct vrna_hc_s vrna_hc_t

    *Typename for the hard constraints data structure vrna_hc_s.*

- typedef struct vrna_hc_up_s vrna_hc_up_t

    *Typename for the single nucleotide hard constraint data structure vrna_hc_up_s.*

- typedef unsigned char() vrna_callback_hc_evaluate(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.*

## Functions

- void vrna_hc_init (vrna_fold_compound_t ∗vc)

    *Initialize/Reset hard constraints to default values.*

- void vrna_hc_add_up (vrna_fold_compound_t ∗vc, int i, unsigned char option)

    *Make a certain nucleotide unpaired.*

- int vrna_hc_add_up_batch (vrna_fold_compound_t ∗vc, vrna_hc_up_t ∗constraints)

    *Apply a list of hard constraints for single nucleotides.*

- void vrna_hc_add_bp (vrna_fold_compound_t ∗vc, int i, int j, unsigned char option)

    *Favorize/Enforce a certain base pair (i,j)*

- void vrna_hc_add_bp_nonspecific (vrna_fold_compound_t ∗vc, int i, int d, unsigned char option)

    *Enforce a nucleotide to be paired (upstream/downstream)*

- void vrna_hc_free (vrna_hc_t ∗hc)

    *Free the memory allocated by a vrna_hc_t data structure.*

- int vrna_hc_add_from_db (vrna_fold_compound_t ∗vc, const char ∗constraint, unsigned int options)

    *Add hard constraints from pseudo dot-bracket notation.*

## 15.10.2 Data Structure Documentation

### 15.10.2.1 struct vrna_hc_s

The hard constraints data structure.

The content of this data structure determines the decomposition pattern used in the folding recursions. Attribute 'matrix' is used as source for the branching pattern of the decompositions during all folding recursions. Any entry in matrix[i,j] consists of the 6 LSB that allows one to distinguish the following types of base pairs:

- in the exterior loop (VRNA_CONSTRAINT_CONTEXT_EXT_LOOP)

- enclosing a hairpin (VRNA_CONSTRAINT_CONTEXT_HP_LOOP)

- enclosing an interior loop (VRNA_CONSTRAINT_CONTEXT_INT_LOOP)

- enclosed by an exterior loop (VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC)

- enclosing a multi branch loop (VRNA_CONSTRAINT_CONTEXT_MB_LOOP)

- enclosed by a multi branch loop (VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC)

The four linear arrays 'up_xxx' provide the number of available unpaired nucleotides (including position i) 3' of each position in the sequence.

**See also**

> vrna_hc_init(), vrna_hc_free(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTE

Collaboration diagram for vrna_hc_s:

**Data Fields**

- int * up_ext

  *A linear array that holds the number of allowed unpaired nucleotides in an exterior loop.*
- int * up_hp

  *A linear array that holds the number of allowed unpaired nucleotides in a hairpin loop.*
- int * up_int

  *A linear array that holds the number of allowed unpaired nucleotides in an interior loop.*
- int * up_ml

  *A linear array that holds the number of allowed unpaired nucleotides in a multi branched loop.*
- vrna_callback_hc_evaluate * f

  *A function pointer that returns whether or not a certain decomposition may be evaluated.*
- void * data

  *A pointer to some structure where the user may store necessary data to evaluate its generic hard constraint function.*
- vrna_callback_free_auxdata * free_data

  *A pointer to a function to free memory occupied by auxiliary data.*
- unsigned char * matrix

  *Upper triangular matrix that encodes where a base pair or unpaired nucleotide is allowed.*

**15.10.2.1.1 Field Documentation**

**15.10.2.1.1.1 free_data**

vrna_callback_free_auxdata* vrna_hc_s::free_data

A pointer to a function to free memory occupied by auxiliary data.

The function this pointer is pointing to will be called upon destruction of the vrna_hc_s, and provided with the vrna_hc_s.data pointer that may hold auxiliary data. Hence, to avoid leaking memory, the user may use this pointer to free memory occupied by auxiliary data.

**15.10.2.2 struct vrna_hc_up_s**

A single hard constraint for a single nucleotide.

**Data Fields**

- int position
    *The sequence position (1-based)*
- unsigned char options
    *The hard constraint option.*

**15.10.3 Macro Definition Documentation**

**15.10.3.1 VRNA_CONSTRAINT_DB**

#define VRNA_CONSTRAINT_DB 16384U

#include <ViennaRNA/constraints_hard.h>

Flag for vrna_constraints_add() to indicate that constraint is passed in pseudo dot-bracket notation.

**See also**

vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

### 15.10.3.2 VRNA_CONSTRAINT_DB_ENFORCE_BP

#define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U

#include <ViennaRNA/constraints_hard.h>

Switch for dot-bracket structure constraint to enforce base pairs.

This flag should be used to really enforce base pairs given in dot-bracket constraint rather than just weakly-enforcing them.

**See also**

> vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

### 15.10.3.3 VRNA_CONSTRAINT_DB_PIPE

#define VRNA_CONSTRAINT_DB_PIPE 65536U

#include <ViennaRNA/constraints_hard.h>

Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the pipe sign '|' (paired with another base)

**See also**

> vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

### 15.10.3.4 VRNA_CONSTRAINT_DB_DOT

#define VRNA_CONSTRAINT_DB_DOT 131072U

#include <ViennaRNA/constraints_hard.h>

dot '.' switch for structure constraints (no constraint at all)

**See also**

> vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

**15.10.3.5   VRNA_CONSTRAINT_DB_X**

```
#define VRNA_CONSTRAINT_DB_X 262144U
```

```
#include <ViennaRNA/constraints_hard.h>
```

'x' switch for structure constraint (base must not pair)

**See also**

>   vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

**15.10.3.6   VRNA_CONSTRAINT_DB_RND_BRACK**

```
#define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U
```

```
#include <ViennaRNA/constraints_hard.h>
```

round brackets '(',')' switch for structure constraint (base i pairs base j)

**See also**

>   vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

**15.10.3.7   VRNA_CONSTRAINT_DB_INTRAMOL**

```
#define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U
```

```
#include <ViennaRNA/constraints_hard.h>
```

Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'l' character (intramolecular pairs only)

**See also**

>   vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

### 15.10.3.8 VRNA_CONSTRAINT_DB_INTERMOL

#define VRNA_CONSTRAINT_DB_INTERMOL 4194304U

#include <ViennaRNA/constraints_hard.h>

Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'e' character (intermolecular pairs only)

**See also**

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

### 15.10.3.9 VRNA_CONSTRAINT_DB_GQUAD

#define VRNA_CONSTRAINT_DB_GQUAD 8388608U

#include <ViennaRNA/constraints_hard.h>

'+' switch for structure constraint (base is involved in a gquad)

**See also**

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

**Warning**

This flag is for future purposes only! No implementation recognizes it yet.

### 15.10.3.10 VRNA_CONSTRAINT_DB_WUSS

#define VRNA_CONSTRAINT_DB_WUSS 33554432U

#include <ViennaRNA/constraints_hard.h>

Flag to indicate Washington University Secondary Structure (WUSS) notation of the hard constraint string.

This secondary structure notation for RNAs is usually used as consensus secondary structure (SS_cons) entry in Stockholm formatted files

**Note**

See

### 15.10.3.11 VRNA_CONSTRAINT_DB_DEFAULT

```
#define VRNA_CONSTRAINT_DB_DEFAULT
```

#include <ViennaRNA/constraints_hard.h>

**Value:**

```
(VRNA_CONSTRAINT_DB \
   | VRNA_CONSTRAINT_DB_PIPE \
   | VRNA_CONSTRAINT_DB_DOT \
   | VRNA_CONSTRAINT_DB_X \
   | VRNA_CONSTRAINT_DB_ANG_BRACK \
   | VRNA_CONSTRAINT_DB_RND_BRACK \
   | VRNA_CONSTRAINT_DB_INTRAMOL \
   | VRNA_CONSTRAINT_DB_INTERMOL \
   | VRNA_CONSTRAINT_DB_GQUAD \
  )
```

Switch for dot-bracket structure constraint with default symbols.

This flag conveniently combines all possible symbols in dot-bracket notation for hard constraints and VRNA_CONSTRAINT_DB

**See also**

> vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

### 15.10.4 Typedef Documentation

#### 15.10.4.1 vrna_callback_hc_evaluate

```
typedef unsigned char() vrna_callback_hc_evaluate(int i, int j, int k, int l, unsigned char d,
void *data)
```

#include <ViennaRNA/constraints_hard.h>

Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.

This is the prototype for callback functions used by the folding recursions to evaluate generic hard constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via vrna_hc_add_data(), or NULL if no auxiliary data was added.

**Notes on Callback Functions**  This callback enables one to over-rule default hard constraints in secondary structure decompositions.

**See also**

> VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML,
> VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM,
> VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM,
> VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM,
> VRNA_DECOMP_EXT_EXT_STEM1, vrna_hc_add_f(), vrna_hc_add_data()

**Parameters**

| | |
|---|---|
| *i* | Left (5') delimiter position of substructure |
| *j* | Right (3') delimiter position of substructure |
| *k* | Left delimiter of decomposition |
| *l* | Right delimiter of decomposition |
| *d* | Decomposition step indicator |
| *data* | Auxiliary data |

**Returns**

A non-zero value if the decomposition is valid, 0 otherwise

### 15.10.5 Function Documentation

#### 15.10.5.1 vrna_hc_init()

```
void vrna_hc_init (
        vrna_fold_compound_t * vc )
```

#include <ViennaRNA/constraints_hard.h>

Initialize/Reset hard constraints to default values.

This function resets the hard constraints to their default values, i.e. all positions may be unpaired in all contexts, and base pairs are allowed in all contexts, if they resemble canonical pairs. Previously set hard constraints will be removed vefore initialization.

**See also**

vrna_hc_add_bp(), vrna_hc_add_bp_nonspecific(), vrna_hc_add_up()

**Parameters**

| | |
|---|---|
| *vc* | The fold compound |

**SWIG Wrapper Notes** This function is attached as method **hc_init()** to objects of type *fold_compound*

#### 15.10.5.2 vrna_hc_add_up()

```
void vrna_hc_add_up (
        vrna_fold_compound_t * vc,
```

```
              int i,
              unsigned char option )
```

#include <ViennaRNA/constraints_hard.h>

Make a certain nucleotide unpaired.

**See also**

> vrna_hc_add_bp(), vrna_hc_add_bp_nonspecific(), vrna_hc_init(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP, VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTE VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

**Parameters**

| | |
|---|---|
| vc | The vrna_fold_compound_t the hard constraints are associated with |
| i | The position that needs to stay unpaired (1-based) |
| option | The options flag indicating how/where to store the hard constraints |

**15.10.5.3   vrna_hc_add_up_batch()**

```
int vrna_hc_add_up_batch (
              vrna_fold_compound_t * vc,
              vrna_hc_up_t * constraints )
```

#include <ViennaRNA/constraints_hard.h>

Apply a list of hard constraints for single nucleotides.

**Parameters**

| | |
|---|---|
| vc | The vrna_fold_compound_t the hard constraints are associated with |
| constraints | The list off constraints to apply, last entry must have position attribute set to 0 |

**15.10.5.4   vrna_hc_add_bp()**

```
void vrna_hc_add_bp (
              vrna_fold_compound_t * vc,
              int i,
              int j,
              unsigned char option )
```

#include <ViennaRNA/constraints_hard.h>

Favorize/Enforce a certain base pair (i,j)

**See also**

vrna_hc_add_bp_nonspecific(), vrna_hc_add_up(), vrna_hc_init(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP,
VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTE
VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC,
VRNA_CONSTRAINT_CONTEXT_ENFORCE, VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

**Parameters**

| vc | The vrna_fold_compound_t the hard constraints are associated with |
|---|---|
| i | The 5' located nucleotide position of the base pair (1-based) |
| j | The 3' located nucleotide position of the base pair (1-based) |
| option | The options flag indicating how/where to store the hard constraints |

**15.10.5.5   vrna_hc_add_bp_nonspecific()**

```
void vrna_hc_add_bp_nonspecific (
            vrna_fold_compound_t * vc,
            int i,
            int d,
            unsigned char option )
```

#include <ViennaRNA/constraints_hard.h>

Enforce a nucleotide to be paired (upstream/downstream)

**See also**

vrna_hc_add_bp(), vrna_hc_add_up(), vrna_hc_init(), VRNA_CONSTRAINT_CONTEXT_EXT_LOOP,
VRNA_CONSTRAINT_CONTEXT_HP_LOOP, VRNA_CONSTRAINT_CONTEXT_INT_LOOP, VRNA_CONSTRAINT_CONTE
VRNA_CONSTRAINT_CONTEXT_MB_LOOP, VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC,
VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

**Parameters**

| vc | The vrna_fold_compound_t the hard constraints are associated with |
|---|---|
| i | The position that needs to stay unpaired (1-based) |
| d | The direction of base pairing ( $d < 0$: pairs upstream, $d > 0$: pairs downstream, $d == 0$: no direction) |
| option | The options flag indicating in which loop type context the pairs may appear |

**15.10.5.6   vrna_hc_free()**

```
void vrna_hc_free (
            vrna_hc_t * hc )
```

#include <ViennaRNA/constraints_hard.h>

Free the memory allocated by a vrna_hc_t data structure.

Use this function to free all memory that was allocated for a data structure of type vrna_hc_t .

**See also**

get_hard_constraints(), vrna_hc_t

**15.10.5.7 vrna_hc_add_from_db()**

```
int vrna_hc_add_from_db (
            vrna_fold_compound_t * vc,
            const char * constraint,
            unsigned int options )
```

```
#include <ViennaRNA/constraints_hard.h>
```

Add hard constraints from pseudo dot-bracket notation.

This function allows one to apply hard constraints from a pseudo dot-bracket notation. The `options` parameter controls, which characters are recognized by the parser. Use the VRNA_CONSTRAINT_DB_DEFAULT convenience macro, if you want to allow all known characters

**See also**

VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_
VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTRAMOL, VRNA_CONSTRAINT_DB_INTERMOL,
VRNA_CONSTRAINT_DB_GQUAD

**Parameters**

| vc | The fold compound |
|---|---|
| constraint | A pseudo dot-bracket notation of the hard constraint. |
| options | The option flags |

**SWIG Wrapper Notes** This function is attached as method **hc_add_from_db()** to objects of type *fold_compound*

## 15.11 Soft constraints

Functions and data structures for secondary structure soft constraints.

### 15.11.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

Soft-constraints are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations. Collaboration diagram for Soft constraints:

### Files

- file constraints_soft.h

    *Functions and data structures for secondary structure soft constraints.*

### Data Structures

- struct vrna_sc_s

    *The soft constraints data structure. More...*

### Typedefs

- typedef struct vrna_sc_s vrna_sc_t

    *Typename for the soft constraints data structure vrna_sc_s.*
- typedef int() vrna_callback_sc_energy(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to retrieve pseudo energy contribution for soft constraint feature.*
- typedef FLT_OR_DBL() vrna_callback_sc_exp_energy(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.*
- typedef vrna_basepair_t ∗() vrna_callback_sc_backtrack(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to retrieve auxiliary base pairs for soft constraint feature.*

### Functions

- void vrna_sc_init (vrna_fold_compound_t ∗vc)

    *Initialize an empty soft constraints data structure within a vrna_fold_compound_t.*
- void vrna_sc_set_bp (vrna_fold_compound_t ∗vc, const FLT_OR_DBL ∗∗constraints, unsigned int options)

    *Set soft constraints for paired nucleotides.*
- void vrna_sc_add_bp (vrna_fold_compound_t ∗vc, int i, int j, FLT_OR_DBL energy, unsigned int options)

    *Add soft constraints for paired nucleotides.*
- void vrna_sc_set_up (vrna_fold_compound_t ∗vc, const FLT_OR_DBL ∗constraints, unsigned int options)

    *Set soft constraints for unpaired nucleotides.*
- void vrna_sc_add_up (vrna_fold_compound_t ∗vc, int i, FLT_OR_DBL energy, unsigned int options)

    *Add soft constraints for unpaired nucleotides.*
- void vrna_sc_remove (vrna_fold_compound_t ∗vc)

*Remove soft constraints from vrna_fold_compound_t.*

- void vrna_sc_free (vrna_sc_t ∗sc)

  *Free memory occupied by a vrna_sc_t data structure.*

- void vrna_sc_add_data (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗free_data)

  *Add an auxiliary data structure for the generic soft constraints callback function.*

- void vrna_sc_add_f (vrna_fold_compound_t ∗vc, vrna_callback_sc_energy ∗f)

  *Bind a function pointer for generic soft constraint feature (MFE version)*

- void vrna_sc_add_bt (vrna_fold_compound_t ∗vc, vrna_callback_sc_backtrack ∗f)

  *Bind a backtracking function pointer for generic soft constraint feature.*

- void vrna_sc_add_exp_f (vrna_fold_compound_t ∗vc, vrna_callback_sc_exp_energy ∗exp_f)

  *Bind a function pointer for generic soft constraint feature (PF version)*

## 15.11.2 Data Structure Documentation

### 15.11.2.1 struct vrna_sc_s

The soft constraints data structure.

Collaboration diagram for vrna_sc_s:

**Data Fields**

- int ∗∗ energy_up

  *Energy contribution for stretches of unpaired nucleotides.*

- FLT_OR_DBL ∗∗ exp_energy_up

  *Boltzmann Factors of the energy contributions for unpaired sequence stretches.*

- int ∗ up_storage

  *Storage container for energy contributions per unpaired nucleotide.*

- vrna_sc_bp_storage_t ∗∗ bp_storage

  *Storage container for energy contributions per base pair.*

- int ∗ energy_stack

  *Pseudo Energy contribution per base pair involved in a stack.*

- FLT_OR_DBL ∗ exp_energy_stack

  *Boltzmann weighted pseudo energy contribution per nucleotide involved in a stack.*

- vrna_callback_sc_energy ∗ f

  *A function pointer used for pseudo energy contribution in MFE calculations.*

- vrna_callback_sc_backtrack ∗ bt

  *A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.*

- vrna_callback_sc_exp_energy ∗ exp_f

  *A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.*

- void ∗ data

  *A pointer to the data object provided for for pseudo energy contribution functions of the generic soft constraints feature.*

- int ∗ energy_bp

  *Energy contribution for base pairs.*

- FLT_OR_DBL ∗ exp_energy_bp

  *Boltzmann Factors of the energy contribution for base pairs.*

- int ∗∗ energy_bp_local

  *Energy contribution for base pairs (sliding window approach)*

- FLT_OR_DBL ∗∗ exp_energy_bp_local

  *Boltzmann Factors of the energy contribution for base pairs (sliding window approach)*

**15.11.2.1.1 Field Documentation**

**15.11.2.1.1.1 f**

vrna_callback_sc_energy* vrna_sc_s::f

A function pointer used for pseudo energy contribution in MFE calculations.

**See also**

vrna_sc_add_f()

**15.11.2.1.1.2 bt**

vrna_callback_sc_backtrack* vrna_sc_s::bt

A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.

**See also**

vrna_sc_add_bt()

**15.11.2.1.1.3 exp_f**

vrna_callback_sc_exp_energy* vrna_sc_s::exp_f

A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.

**See also**

vrna_sc_add_exp_f()

**15.11.3 Typedef Documentation**

### 15.11.3.1 vrna_callback_sc_energy

```
typedef int() vrna_callback_sc_energy(int i, int j, int k, int l, unsigned char d, void *data)
```

```
#include <ViennaRNA/constraints_soft.h>
```

Callback to retrieve pseudo energy contribution for soft constraint feature.

This is the prototype for callback functions used by the folding recursions to evaluate generic soft constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via vrna_sc_add_data(), or NULL if no auxiliary data was added.

**Notes on Callback Functions** This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure.

**See also**

> VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_bt(), vrna_sc_add_data()

**Parameters**

| | |
|---|---|
| *i* | Left (5') delimiter position of substructure |
| *j* | Right (3') delimiter position of substructure |
| *k* | Left delimiter of decomposition |
| *l* | Right delimiter of decomposition |
| *d* | Decomposition step indicator |
| *data* | Auxiliary data |

**Returns**

Pseudo energy contribution in deka-kalories per mol

**15.11.3.2 vrna_callback_sc_exp_energy**

```
typedef FLT_OR_DBL() vrna_callback_sc_exp_energy(int i, int j, int k, int l, unsigned char d,
void *data)
```

#include <ViennaRNA/constraints_soft.h>

Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.

This is the prototype for callback functions used by the partition function recursions to evaluate generic soft constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via vrna_sc_add_data(), or NULL if no auxiliary data was added.

**Notes on Callback Functions** This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure (Partition function variant, i.e. contributions must be returned as Boltzmann factors).

**See also**

VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_sc_add_exp_f(), vrna_sc_add_f(), vrna_sc_add_bt(), vrna_sc_add_data()

**Parameters**

| | |
|---|---|
| *i* | Left (5') delimiter position of substructure |
| *j* | Right (3') delimiter position of substructure |
| *k* | Left delimiter of decomposition |
| *l* | Right delimiter of decomposition |
| *d* | Decomposition step indicator |
| *data* | Auxiliary data |

**Returns**

Pseudo energy contribution in deka-kalories per mol

### 15.11.3.3 vrna_callback_sc_backtrack

```
typedef vrna_basepair_t*() vrna_callback_sc_backtrack(int i, int j, int k, int l, unsigned
char d, void *data)
```

```
#include <ViennaRNA/constraints_soft.h>
```

Callback to retrieve auxiliary base pairs for soft constraint feature.

**Notes on Callback Functions** This callback enables one to add auxiliary base pairs in the backtracking steps of hairpin- and interior loops.

**See also**

VRNA_DECOMP_PAIR_HP, VRNA_DECOMP_PAIR_IL, VRNA_DECOMP_PAIR_ML, VRNA_DECOMP_ML_ML_ML, VRNA_DECOMP_ML_STEM, VRNA_DECOMP_ML_ML, VRNA_DECOMP_ML_UP, VRNA_DECOMP_ML_ML_STEM, VRNA_DECOMP_ML_COAXIAL, VRNA_DECOMP_EXT_EXT, VRNA_DECOMP_EXT_UP, VRNA_DECOMP_EXT_STEM, VRNA_DECOMP_EXT_EXT_EXT, VRNA_DECOMP_EXT_STEM_EXT, VRNA_DECOMP_EXT_EXT_STEM, VRNA_DECOMP_EXT_EXT_STEM1, vrna_sc_add_bt(), vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_data()

**Parameters**

| | |
|---|---|
| *i* | Left (5') delimiter position of substructure |
| *j* | Right (3') delimiter position of substructure |
| *k* | Left delimiter of decomposition |
| *l* | Right delimiter of decomposition |
| *d* | Decomposition step indicator |
| *data* | Auxiliary data |

**Returns**

List of additional base pairs

### 15.11.4 Function Documentation

#### 15.11.4.1 vrna_sc_init()

```
void vrna_sc_init (
            vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/constraints_soft.h>
```

Initialize an empty soft constraints data structure within a vrna_fold_compound_t.

This function adds a proper soft constraints data structure to the vrna_fold_compound_t data structure. If soft constraints already exist within the fold compound, they are removed.

**Note**

Accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE and VRNA_FC_TYPE_COMPARATIVE

**See also**

vrna_sc_set_bp(), vrna_sc_set_up(), vrna_sc_add_SHAPE_deigan(), vrna_sc_add_SHAPE_zarringhalam(), vrna_sc_remove(), vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_pre(), vrna_sc_add_post()

**Parameters**

| vc | The vrna_fold_compound_t where an empty soft constraint feature is to be added to |
|----|-----------------------------------------------------------------------------------|

**SWIG Wrapper Notes** This function is attached as method **sc_init()** to objects of type *fold_compound*

**15.11.4.2 vrna_sc_set_bp()**

```
void vrna_sc_set_bp (
            vrna_fold_compound_t * vc,
            const FLT_OR_DBL ** constraints,
            unsigned int options )
```

```
#include <ViennaRNA/constraints_soft.h>
```

Set soft constraints for paired nucleotides.

**Note**

This function replaces any pre-exisitng soft constraints with the ones supplied in `constraints`.

**See also**

vrna_sc_add_bp(), vrna_sc_set_up(), vrna_sc_add_up()

**Parameters**

| vc | The vrna_fold_compound_t the soft constraints are associated with |
|-------------|------------------------------------------------------------------|
| constraints | A two-dimensional array of pseudo free energies in $kcal/mol$ |
| options | The options flag indicating how/where to store the soft constraints |

**SWIG Wrapper Notes** This function is attached as method **sc_set_bp()** to objects of type *fold_compound*

**15.11.4.3 vrna_sc_add_bp()**

```
void vrna_sc_add_bp (
            vrna_fold_compound_t * vc,
            int i,
            int j,
            FLT_OR_DBL energy,
            unsigned int options )
```

#include <ViennaRNA/constraints_soft.h>

Add soft constraints for paired nucleotides.

**See also**

vrna_sc_set_bp(), vrna_sc_set_up(), vrna_sc_add_up()

**Parameters**

| vc | The vrna_fold_compound_t the soft constraints are associated with |
|---|---|
| i | The 5' position of the base pair the soft constraint is added for |
| j | The 3' position of the base pair the soft constraint is added for |
| energy | The free energy (soft-constraint) in $kcal/mol$ |
| options | The options flag indicating how/where to store the soft constraints |

**SWIG Wrapper Notes** This function is attached as an overloaded method **sc_add_bp()** to objects of type *fold←_compound*. The method either takes arguments for a single base pair (i,j) with the corresponding energy value:

```
fold_compound.sc_add_bp(i, j, energy, options)
```

or an entire 2-dimensional matrix with dimensions n x n that stores free energy contributions for any base pair (i,j) with $1 \leq i < j \leq n$:

```
fold_compound.sc_add_bp(matrix, options)
```

In both variants, the `options` argument is optional can may be omitted.

**15.11.4.4 vrna_sc_set_up()**

```
void vrna_sc_set_up (
            vrna_fold_compound_t * vc,
            const FLT_OR_DBL * constraints,
            unsigned int options )
```

#include <ViennaRNA/constraints_soft.h>

Set soft constraints for unpaired nucleotides.

**Note**

> This function replaces any pre-exisitng soft constraints with the ones supplied in `constraints`.

**See also**

> vrna_sc_add_up(), vrna_sc_set_bp(), vrna_sc_add_bp()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t the soft constraints are associated with |
| *constraints* | A vector of pseudo free energies in $kcal/mol$ |
| *options* | The options flag indicating how/where to store the soft constraints |

**SWIG Wrapper Notes**  This function is attached as method **sc_set_up()** to objects of type *fold_compound*

**15.11.4.5  vrna_sc_add_up()**

```
void vrna_sc_add_up (
            vrna_fold_compound_t * vc,
            int i,
            FLT_OR_DBL energy,
            unsigned int options )
```

`#include <ViennaRNA/constraints_soft.h>`

Add soft constraints for unpaired nucleotides.

**See also**

> vrna_sc_set_up(), vrna_sc_add_bp(), vrna_sc_set_bp()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t the soft constraints are associated with |
| *i* | The nucleotide position the soft constraint is added for |
| *energy* | The free energy (soft-constraint) in $kcal/mol$ |
| *options* | The options flag indicating how/where to store the soft constraints |

**SWIG Wrapper Notes**  This function is attached as an overloaded method **sc_add_up()** to objects of type *fold↩_compound*. The method either takes arguments for a single nucleotide $i$ with the corresponding energy value:

```
fold_compound.sc_add_up(i, energy, options)
```

or an entire vector that stores free energy contributions for each nucleotide $i$ with $1 \leq i \leq n$:

```
                        fold_compound.sc_add_bp(vector, options)
```

In both variants, the `options` argument is optional can may be omitted.

**15.11.4.6 vrna_sc_remove()**

```
void vrna_sc_remove (
            vrna_fold_compound_t * vc )
```

`#include <ViennaRNA/constraints_soft.h>`

Remove soft constraints from vrna_fold_compound_t.

**Note**

Accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE and VRNA_FC_TYPE_COMPARATIVE

**Parameters**

| vc | The vrna_fold_compound_t possibly containing soft constraints |
|---|---|

**SWIG Wrapper Notes** This function is attached as method **sc_remove()** to objects of type *fold_compound*

**15.11.4.7 vrna_sc_free()**

```
void vrna_sc_free (
            vrna_sc_t * sc )
```

`#include <ViennaRNA/constraints_soft.h>`

Free memory occupied by a vrna_sc_t data structure.

**Parameters**

| sc | The data structure to free from memory |
|---|---|

**15.11.4.8 vrna_sc_add_data()**

```
void vrna_sc_add_data (
            vrna_fold_compound_t * vc,
```

```
        void * data,
        vrna_callback_free_auxdata * free_data )
```

#include <ViennaRNA/constraints_soft.h>

Add an auxiliary data structure for the generic soft constraints callback function.

**See also**

vrna_sc_add_f(), vrna_sc_add_exp_f(), vrna_sc_add_bt()

**Parameters**

| vc | The fold compound the generic soft constraint function should be bound to |
|---|---|
| data | A pointer to the data structure that holds required data for function 'f' |
| free_data | A pointer to a function that free's the memory occupied by data (Maybe NULL) |

**SWIG Wrapper Notes** This function is attached as method **sc_add_data()** to objects of type *fold_compound*

**15.11.4.9 vrna_sc_add_f()**

```
void vrna_sc_add_f (
        vrna_fold_compound_t * vc,
        vrna_callback_sc_energy * f )
```

#include <ViennaRNA/constraints_soft.h>

Bind a function pointer for generic soft constraint feature (MFE version)

This function allows one to easily bind a function pointer and corresponding data structure to the soft constraint part vrna_sc_t of the vrna_fold_compound_t. The function for evaluating the generic soft constraint feature has to return a pseudo free energy $\hat{E}$ in $dacal/mol$, where $1dacal/mol = 10cal/mol$.

**See also**

vrna_sc_add_data(), vrna_sc_add_bt(), vrna_sc_add_exp_f()

**Parameters**

| vc | The fold compound the generic soft constraint function should be bound to |
|---|---|
| f | A pointer to the function that evaluates the generic soft constraint feature |

**SWIG Wrapper Notes** This function is attached as method **sc_add_f()** to objects of type *fold_compound*

**15.11.4.10 vrna_sc_add_bt()**

```
void vrna_sc_add_bt (
            vrna_fold_compound_t * vc,
            vrna_callback_sc_backtrack * f )
```

#include <ViennaRNA/constraints_soft.h>

Bind a backtracking function pointer for generic soft constraint feature.

This function allows one to easily bind a function pointer to the soft constraint part vrna_sc_t of the vrna_fold_compound_t. The provided function should be used for backtracking purposes in loop regions that were altered via the generic soft constraint feature. It has to return an array of vrna_basepair_t data structures, were the last element in the list is indicated by a value of -1 in it's i position.

**See also**

vrna_sc_add_data(), vrna_sc_add_f(), vrna_sc_add_exp_f()

**Parameters**

| vc | The fold compound the generic soft constraint function should be bound to |
|---|---|
| f | A pointer to the function that returns additional base pairs |

**SWIG Wrapper Notes** This function is attached as method **sc_add_bt()** to objects of type *fold_compound*

**15.11.4.11 vrna_sc_add_exp_f()**

```
void vrna_sc_add_exp_f (
            vrna_fold_compound_t * vc,
            vrna_callback_sc_exp_energy * exp_f )
```

#include <ViennaRNA/constraints_soft.h>

Bind a function pointer for generic soft constraint feature (PF version)

This function allows one to easily bind a function pointer and corresponding data structure to the soft constraint part vrna_sc_t of the vrna_fold_compound_t. The function for evaluating the generic soft constraint feature has to return a pseudo free energy $\hat{E}$ as Boltzmann factor, i.e. $exp(-\hat{E}/kT)$. The required unit for $E$ is $cal/mol$.

**See also**

vrna_sc_add_bt(), vrna_sc_add_f(), vrna_sc_add_data()

**Parameters**

| vc | The fold compound the generic soft constraint function should be bound to |
|---|---|
| exp↩ _f | A pointer to the function that evaluates the generic soft constraint feature |

**SWIG Wrapper Notes**  This function is attached as method **sc_add_exp_f()** to objects of type *fold_compound*

## 15.12 The RNA secondary structure landscape

### 15.12.1 Detailed Description

Collaboration diagram for The RNA secondary structure landscape:

**Modules**

- Neighborhood relation and move sets for secondary structures

  *Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.*
- Refolding paths of secondary structures

## 15.13 Minimum Free Energy (MFE) algorithms

### 15.13.1 Detailed Description

Collaboration diagram for Minimum Free Energy (MFE) algorithms:

**Modules**

- MFE Consensus Structures for Sequence Alignment(s)
- Calculating MFE representatives of a Distance Based Partitioning

    *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.*

- MFE Structures of single Nucleic Acid Sequences

    *This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.*

- MFE Structures of two hybridized Sequences

    *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.*

- Local MFE Structure Prediction and Z-scores

**Files**

- file fold.h

    *MFE calculations for single RNA sequences.*

- file mfe.h

    *Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.*

**Typedefs**

- typedef void() vrna_mfe_window_callback(int start, int end, const char ∗structure, float en, void ∗data)

    *The default callback for sliding window MFE structure predictions.*

**Functions**

- float vrna_mfe (vrna_fold_compound_t ∗vc, char ∗structure)

    *Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.*

### 15.13.2 Typedef Documentation

**15.13.2.1 vrna_mfe_window_callback**

```
typedef void() vrna_mfe_window_callback(int start, int end, const char *structure, float en,
void *data)
```

`#include <`VriennaRNA/mfe.h`>`

The default callback for sliding window MFE structure predictions.

**Notes on Callback Functions** This function will be called for each hit in a sliding window MFE prediction.

**See also**

vrna_mfe_window()

**Parameters**

| *start* | provides the first position of the hit (1-based, relative to entire sequence/alignment) |
|---|---|
| *end* | provides the last position of the hit (1-based, relative to the entire sequence/alignment) |
| *structure* | provides the (sub)structure in dot-bracket notation |
| *en* | is the free energy of the structure hit in kcal/mol |
| *data* | is some arbitrary data pointer passed through by the function executing the callback |

**15.13.3 Function Documentation**

**15.13.3.1 vrna_mfe()**

```
float vrna_mfe (
            vrna_fold_compound_t * vc,
            char * structure )
```

`#include <`VriennaRNA/mfe.h`>`

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

Depending on the type of the provided vrna_fold_compound_t, this function predicts the MFE for a single sequence, or a corresponding averaged MFE for a sequence alignment. If backtracking is activated, it also constructs the corresponding secondary structure, or consensus structure. Therefore, the second parameter, *structure*, has to point to an allocated block of memory with a size of at least $\mathrm{strlen}(\mathrm{sequence}) + 1$ to store the backtracked MFE structure. (For consensus structures, this is the length of the alignment + 1. If `NULL` is passed, no backtracking will be performed.

**Note**

This function is polymorphic. It accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE, and VRNA_FC_TYPE_COMPARATIVE.

**See also**

vrna_fold_compound_t, vrna_fold_compound(), vrna_fold(), vrna_circfold(), vrna_fold_compound_comparative(), vrna_alifold(), vrna_circalifold()

**Parameters**

| | |
|---|---|
| *vc* | fold compound |
| *structure* | A pointer to the character array where the secondary structure in dot-bracket notation will be written to (Maybe NULL) |

**Returns**

the minimum free energy (MFE) in kcal/mol

**SWIG Wrapper Notes**  This function is attached as method **mfe()** to objects of type *fold_compound*

## 15.14 Partition function and equilibrium properties

Compute the partition function and various equilibrium properties derived from it.

### 15.14.1 Detailed Description

Compute the partition function and various equilibrium properties derived from it.

Collaboration diagram for Partition function and equilibrium properties:

### Modules

- Partition Function and Base Pair Probabilities for Sequence Alignment(s)
- Calculate Partition Functions of a Distance Based Partitioning

    *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.*
- Partition Function for two hybridized Sequences

    *Partition Function Cofolding.*
- Partition Function for two hybridized Sequences as a stepwise Process

    *RNA-RNA interaction as a stepwise process.*
- Partition Function and Equilibrium Probabilities for Locally Stable Secondary Structures

### Files

- file concentrations.h

    *Concentration computations for RNA-RNA interactions.*
- file equilibrium_probs.h

    *Equilibrium Probability implementations.*
- file part_func.h

    *Partition function implementations.*
- file part_func_co.h

    *Partition function for two RNA sequences.*
- file part_func_up.h

    *Implementations for accessibility and RNA-RNA interaction as a stepwise process.*

### Functions

- double vrna_mean_bp_distance_pr (int length, FLT_OR_DBL ∗pr)

    *Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.*
- double vrna_mean_bp_distance (vrna_fold_compound_t ∗vc)

    *Get the mean base pair distance in the thermodynamic ensemble.*
- vrna_ep_t ∗ vrna_stack_prob (vrna_fold_compound_t ∗vc, double cutoff)

    *Compute stacking probabilities.*
- float vrna_pf (vrna_fold_compound_t ∗vc, char ∗structure)

    *Compute the partition function $Q$ for a given RNA sequence, or sequence alignment.*
- float vrna_pf_fold (const char ∗sequence, char ∗structure, vrna_ep_t ∗∗pl)

*Compute Partition function $Q$ (and base pair probabilities) for an RNA sequence using a comparative method.*

- float vrna_pf_circfold (const char *sequence, char *structure, vrna_ep_t **pl)

  *Compute Partition function $Q$ (and base pair probabilities) for a circular RNA sequences using a comparative method.*

- float pf_fold_par (const char *sequence, char *structure, vrna_exp_param_t *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, vrna_exp_param_t *parameters)

  *Recalculate energy parameters.*

- FLT_OR_DBL * 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.*

- int get_pf_arrays (short **S_p, short **S1_p, char **ptype_p, FLT_OR_DBL **qb_p, FLT_OR_DBL **qm↩ _p, FLT_OR_DBL **q1k_p, FLT_OR_DBL **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, FLT_OR_DBL *pr)

  *Get the mean base pair distance in the thermodynamic ensemble.*

- vrna_ep_t * vrna_plist_from_probs (vrna_fold_compound_t *vc, double cut_off)

  *Create a vrna_ep_t from base pair probability matrix.*

- void assign_plist_from_pr (vrna_ep_t **pl, FLT_OR_DBL *probs, int length, double cutoff)

  *Create a vrna_ep_t from a probability matrix.*

## 15.14.2 Function Documentation

### 15.14.2.1 vrna_mean_bp_distance_pr()

```
double vrna_mean_bp_distance_pr (
            int length,
            FLT_OR_DBL * pr )
```

`#include <ViennaRNA/equilibrium_probs.h>`

Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.

$< d >= \sum_{a,b} p_a p_b d(S_a, S_b)$
this can be computed from the pair probs $p_ij$ as
$< d >= \sum_{ij} p_{ij}(1 - p_{ij})$

**Parameters**

| length | The length of the sequence |
|--------|----------------------------|
| pr | The matrix containing the base pair probabilities |

**Returns**

The mean pair distance of the structure ensemble

**15.14.2.2 vrna_mean_bp_distance()**

```
double vrna_mean_bp_distance (
              vrna_fold_compound_t * vc )
```

`#include <ViennaRNA/equilibrium_probs.h>`

Get the mean base pair distance in the thermodynamic ensemble.

$< d >= \sum_{a,b} p_a p_b d(S_a, S_b)$
this can be computed from the pair probs $p_i j$ as
$< d >= \sum_{ij} p_{ij}(1 - p_{ij})$

**Parameters**

| vc | The fold compound data structure |
|----|----------------------------------|

**Returns**

The mean pair distance of the structure ensemble

**SWIG Wrapper Notes** This function is attached as method **mean_bp_distance()** to objects of type *fold_compound*

**15.14.2.3 vrna_stack_prob()**

```
vrna_ep_t* vrna_stack_prob (
              vrna_fold_compound_t * vc,
              double cutoff )
```

`#include <ViennaRNA/equilibrium_probs.h>`

Compute stacking probabilities.

For each possible base pair $(i, j)$, compute the probability of a stack $(i, j)$, $(i + 1, j - 1)$.

**Parameters**

| | |
|---|---|
| *vc* | The fold compound data structure with precomputed base pair probabilities |
| *cutoff* | A cutoff value that limits the output to stacks with $p > \mathrm{cutoff}$. |

**Returns**

A list of stacks with enclosing base pair $(i, j)$ and probabiltiy $p$

**15.14.2.4  vrna_pf()**

```
float vrna_pf (
            vrna_fold_compound_t * vc,
            char * structure )
```

`#include <ViennaRNA/part_func.h>`

Compute the partition function $Q$ for a given RNA sequence, or sequence alignment.

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 the 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**

This function is polymorphic. It accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE, and VRNA_FC_TYPE_COMPARATIVE.

**See also**

vrna_fold_compound_t, vrna_fold_compound(), vrna_pf_fold(), vrna_pf_circfold(), vrna_fold_compound_comparative(), vrna_pf_alifold(), vrna_pf_circalifold(), vrna_db_from_probs(), vrna_exp_params(), vrna_aln_pinfo()

**Parameters**

| | | |
|---|---|---|
| in,out | *vc* | The fold compound data structure |
| in,out | *structure* | A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL) |

**Returns**

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

**SWIG Wrapper Notes**  This function is attached as method **pf()** to objects of type *fold_compound*

**15.14.2.5 vrna_pf_fold()**

```
float vrna_pf_fold (
            const char * sequence,
            char * structure,
            vrna_ep_t ** pl )
```

#include <ViennaRNA/part_func.h>

Compute Partition function $Q$ (and base pair probabilities) for an RNA sequence using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for an RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_pf_circfold(), vrna_pf(), vrna_fold_compound(), vrna_fold_compound_t

**Parameters**

| sequence | RNA sequence |
|---|---|
| structure | A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL) |
| pl | A pointer to a list of vrna_ep_t to store pairing probabilities (Maybe NULL) |

**Returns**

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

**15.14.2.6 vrna_pf_circfold()**

```
float vrna_pf_circfold (
            const char * sequence,
            char * structure,
            vrna_ep_t ** pl )
```

#include <ViennaRNA/part_func.h>

Compute Partition function $Q$ (and base pair probabilities) for a circular RNA sequences using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for a circular RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

**See also**

> vrna_pf_fold(), vrna_pf(), vrna_fold_compound(), vrna_fold_compound_t

**Parameters**

| sequence | A circular RNA sequence |
|---|---|
| structure | A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL) |
| pl | A pointer to a list of vrna_ep_t to store pairing probabilities (Maybe NULL) |

**Returns**

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

**15.14.2.7 pf_fold_par()**

```
float pf_fold_par (
            const char * sequence,
            char * structure,
            vrna_exp_param_t * parameters,
            int calculate_bppm,
            int is_constrained,
            int is_circular )
```

#include <ViennaRNA/part_func.h>

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 the 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.

**Deprecated** Use vrna_pf() instead

**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**

> vrna_pf(), bppm_to_structure(), export_bppm(), vrna_exp_params(), free_pf_arrays()

**Parameters**

| in | *sequence* | The RNA sequence input |
|---|---|---|
| in,out | *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) |
| in | *parameters* | Data structure containing the precalculated Boltzmann factors |
| in | *calculate_bppm* | Switch to Base pair probability calculations on/off (0==off) |
| in | *is_constrained* | Switch to indicate that a structure contraint is passed via the structure argument (0==off) |
| in | *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

**15.14.2.8 pf_fold()**

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

`#include <ViennaRNA/part_func.h>`

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

**15.14.2.9 pf_circ_fold()**

```
float pf_circ_fold (
            const char * sequence,
            char * structure )
```

#include <ViennaRNA/part_func.h>

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**

vrna_pf()

**Deprecated** Use vrna_pf() instead!

**Parameters**

| in | *sequence* | The RNA sequence input |
|---|---|---|
| in,out | *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

**15.14.2.10 free_pf_arrays()**

```
void free_pf_arrays (
            void  )
```

`#include <ViennaRNA/part_func.h>`

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

**Deprecated** See vrna_fold_compound_t and its related functions for how to free memory occupied by the dynamic programming matrices

**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()

**15.14.2.11 update_pf_params()**

```
void update_pf_params (
            int length )
```

#include <ViennaRNA/part_func.h>

Recalculate energy parameters.

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

**Deprecated** Use vrna_exp_params_subst() instead

**15.14.2.12 update_pf_params_par()**

```
void update_pf_params_par (
            int length,
            vrna_exp_param_t * parameters )
```

#include <ViennaRNA/part_func.h>

Recalculate energy parameters.

**Deprecated** Use vrna_exp_params_subst() instead

**15.14.2.13 export_bppm()**

```
FLT_OR_DBL* export_bppm (
            void  )
```

#include <ViennaRNA/part_func.h>

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(), vrna_idx_row_wise()

**Returns**

    A pointer to the base pair probability array

**15.14.2.14 get_pf_arrays()**

```
int get_pf_arrays (
            short ** S_p,
            short ** S1_p,
            char ** ptype_p,
            FLT_OR_DBL ** qb_p,
            FLT_OR_DBL ** qm_p,
            FLT_OR_DBL ** q1k_p,
            FLT_OR_DBL ** qln_p )
```

`#include <ViennaRNA/part_func.h>`

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**

| out | S_p | A pointer to the 'S' array (integer representation of nucleotides) |
|-----|------|-------------------------------------------------------------------|
| out | S1_p | A pointer to the 'S1' array (2nd integer representation of nucleotides) |
| out | ptype↩ _p | A pointer to the pair type matrix |
| out | qb_p | A pointer to the $Q^B$ matrix |
| out | qm_p | A pointer to the $Q^M$ matrix |
| out | q1k_p | A pointer to the 5' slice of the Q matrix ( $q1k(k) = Q(1, k)$ ) |
| out | 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

**15.14.2.15 mean_bp_distance()**

```
double mean_bp_distance (
            int length )
```
`#include <ViennaRNA/part_func.h>`

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

**Deprecated** Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

**See also**

vrna_mean_bp_distance(), vrna_mean_bp_distance_pr()

**Parameters**

| *length* | |
|---|---|

**Returns**

mean base pair distance in thermodynamic ensemble

**15.14.2.16   mean_bp_distance_pr()**

```
double mean_bp_distance_pr (
            int length,
            FLT_OR_DBL * pr )
```

#include <ViennaRNA/part_func.h>

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

**Deprecated**  Use vrna_mean_bp_distance() or vrna_mean_bp_distance_pr() instead!

**Parameters**

| *length* | The length of the sequence |
|---|---|
| *pr* | The matrix containing the base pair probabilities |

**Returns**

The mean pair distance of the structure ensemble

**15.14.2.17   vrna_plist_from_probs()**

```
vrna_ep_t* vrna_plist_from_probs (
            vrna_fold_compound_t * vc,
            double cut_off )
```

#include <ViennaRNA/structure_utils.h>

Create a vrna_ep_t from base pair probability matrix.

The probability matrix provided via the vrna_fold_compound_t 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

**Parameters**

| in | *vc* | The fold compound |
|----|------|-------------------|
| in | *cut_off* | The cutoff value |

**Returns**

A pointer to the plist that is to be created

**15.14.2.18   assign_plist_from_pr()**

```
void assign_plist_from_pr (
            vrna_ep_t ** pl,
            FLT_OR_DBL * probs,
            int length,
            double cutoff )
```

`#include <`ViennaRNA/structure_utils.h`>`

Create a vrna_ep_t 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

**Deprecated**  Use vrna_plist_from_probs() instead!

**Parameters**

| out | *pl* | A pointer to the vrna_ep_t that is to be created |
|-----|------|--------------------------------------------------|
| in | *probs* | The probability matrix used for creating the plist |
| in | *length* | The length of the RNA sequence |
| in | *cutoff* | The cutoff value |

## 15.15 Suboptimals and representative structures

Sample and enumerate suboptimal secondary structures from RNA sequence data.

### 15.15.1 Detailed Description

Sample and enumerate suboptimal secondary structures from RNA sequence data.

Collaboration diagram for Suboptimals and representative structures:

### Modules

- Suboptimal structures sensu Stiegler et al. 1984 / Zuker et al. 1989
- Suboptimal structures within an energy band arround the MFE
- Structure sampling from the ensemble
- Compute the structure with maximum expected accuracy (MEA)
- Compute the centroid structure

### Files

- file boltzmann_sampling.h

  *Boltzmann Sampling of secondary structures from the ensemble.*
- file centroid.h

  *Centroid structure computation.*
- file MEA.h

  *Computes a MEA (maximum expected accuracy) structure.*
- file mm.h

  *Several Maximum Matching implementations.*
- file subopt.h

  *RNAsubopt and density of states declarations.*

## 15.16 Suboptimal structures sensu Stiegler et al. 1984 / Zuker et al. 1989

### 15.16.1 Detailed Description

Collaboration diagram for Suboptimal structures sensu Stiegler et al. 1984 / Zuker et al. 1989:

### Functions

- vrna_subopt_solution_t ∗ vrna_subopt_zuker (vrna_fold_compound_t ∗vc)

  *Compute Zuker type suboptimal structures.*
- SOLUTION ∗ zukersubopt (const char ∗string)

  *Compute Zuker type suboptimal structures.*
- SOLUTION ∗ zukersubopt_par (const char ∗string, vrna_param_t ∗parameters)

  *Compute Zuker type suboptimal structures.*

### 15.16.2 Function Documentation

#### 15.16.2.1 vrna_subopt_zuker()

```
vrna_subopt_solution_t * vrna_subopt_zuker (
            vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker [23] , 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.

**Note**

> This function internally uses the cofold implementation to compute the suboptimal structures. For that purpose, the function doubles the sequence and enlarges the DP matrices, which in fact will grow by a factor of 4 during the computation! At the end of the structure prediction, everything will be re-set to its original requriements, i.e. normal sequence, normal (empty) DP matrices.

**Bug** Due to resizing, any pre-existing constraints will be lost!

**See also**

> vrna_subopt(), zukersubopt(), zukersubopt_par()

**Parameters**

| | |
|---|---|
| *vc* | fold compound |

**Returns**

List of zuker suboptimal structures

**SWIG Wrapper Notes** This function is attached as method **subopt_zuker()** to objects of type *fold_compound*

**15.16.2.2 zukersubopt()**

SOLUTION* zukersubopt (
            const char * *string* )

#include <ViennaRNA/subopt.h>

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.

**Deprecated** use vrna_zukersubopt() instead

**Parameters**

| | |
|---|---|
| *string* | RNA sequence |

**Returns**

List of zuker suboptimal structures

**15.16.2.3 zukersubopt_par()**

SOLUTION* zukersubopt_par (
            const char * *string,*
            vrna_param_t * *parameters* )

#include <ViennaRNA/subopt.h>

Compute Zuker type suboptimal structures.

**Deprecated** use vrna_zukersubopt() instead

## 15.17 Suboptimal structures within an energy band arround the MFE

### 15.17.1 Detailed Description

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

**Typedefs**

- typedef void() vrna_subopt_callback(const char ∗stucture, float energy, void ∗data)

    *Callback for vrna_subopt_cb()*

**Functions**

- vrna_subopt_solution_t ∗ vrna_subopt (vrna_fold_compound_t ∗vc, int delta, int sorted, FILE ∗fp)

    *Returns list of subopt structures or writes to fp.*

- void vrna_subopt_cb (vrna_fold_compound_t ∗vc, int delta, vrna_subopt_callback ∗cb, void ∗data)

    *Generate suboptimal structures within an energy band arround the MFE.*

- 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, vrna_param_t ∗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**

- double print_energy

    *printing threshold for use with logML*

- int subopt_sorted

    *Sort output by energy.*

### 15.17.2 Typedef Documentation

#### 15.17.2.1 vrna_subopt_callback

```
typedef void() vrna_subopt_callback(const char *stucture, float energy, void *data)
```

```
#include <ViennaRNA/subopt.h>
```

Callback for vrna_subopt_cb()

**Notes on Callback Functions** This function will be called for each suboptimal secondary structure that is success-fully backtraced.

**See also**

vrna_subopt_cb()

**Parameters**

| structure | The suboptimal secondary structure in dot-bracket notation |
|---|---|
| energy | The free energy of the secondary structure in kcal/mol |
| data | Some arbitrary, auxiliary data address as passed to vrna_subopt_cb() |

### 15.17.3 Function Documentation

#### 15.17.3.1 vrna_subopt()

```
vrna_subopt_solution_t * vrna_subopt (
            vrna_fold_compound_t * vc,
            int delta,
            int sorted,
            FILE * fp )
```

```
#include <ViennaRNA/subopt.h>
```

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, see [21]. The results are either directly written to a 'fp' (if 'fp' is not NULL), or (fp==NULL) returned in a vrna_subopt_solution_t ∗ list terminated by an entry were the 'structure' member is NULL.

**Note**

> This function requires all multibranch loop DP matrices for unique multibranch loop backtracing. Therefore, the supplied vrna_fold_compound_t vc (argument 1) must be initialized with vrna_md_t.uniq_ML = 1, for instance like this:
>
> ```
> vrna_md_t md;
> vrna_md_set_default(&md);
> md.uniq_ML = 1;
>
> vrna_fold_compound_t *vc=vrna_fold_compound("GGGGGGAAAAAACCCCCC", &md
>       , VRNA_OPTION_DEFAULT);
> ```

**See also**

> vrna_subopt_cb(), vrna_subopt_zuker()

**Parameters**

| vc | |
|---|---|
| delta | |
| sorted | Sort results by energy in ascending order |
| fp | |

**Returns**

**SWIG Wrapper Notes** This function is attached as method **subopt()** to objects of type *fold_compound*

**15.17.3.2 vrna_subopt_cb()**

```
void vrna_subopt_cb (
            vrna_fold_compound_t * vc,
            int delta,
            vrna_subopt_callback * cb,
            void * data )
```

```
#include <ViennaRNA/subopt.h>
```

Generate suboptimal structures within an energy band arround the MFE.

This is the most generic implementation of the suboptimal structure generator according to Wuchty et al. 1999 [21]. Identical to vrna_subopt(), it computes all secondary structures within an energy band `delta` arround the MFE. However, this function does not print the resulting structures and their corresponding free energies to a file pointer, or returns them as a list. Instead, it calls a user-provided callback function which it passes the structure in dot-bracket format, the corresponding free energy in kcal/mol, and a user-provided data structure each time a structure was backtracked successfully. This function indicates the final output, i.e. the end of the backtracking procedure by passing NULL instead of an actual dot-bracket string to the callback.

**Note**

This function requires all multibranch loop DP matrices for unique multibranch loop backtracing. Therefore, the supplied vrna_fold_compound_t vc (argument 1) must be initialized with vrna_md_t.uniq_ML = 1, for instance like this:

```
vrna_md_t md;
vrna_md_set_default(&md);
md.uniq_ML = 1;

vrna_fold_compound_t *vc=vrna_fold_compound("GGGGGGAAAAAACCCCCC", &md
    , VRNA_OPTION_DEFAULT);
```

**See also**

vrna_subopt_callback, vrna_subopt(), vrna_subopt_zuker()

**Parameters**

| vc | fold compount with the sequence data |
| --- | --- |
| delta | Energy band around the MFE in 10cal/mol, i.e. deka-calories |
| cb | Pointer to a callback function that handles the backtracked structure and its free energy in kcal/mol |
| data | Pointer to some data structure that is passed along to the callback |

**SWIG Wrapper Notes** This function is attached as method **subopt_cb()** to objects of type *fold_compound*

**15.17.3.3 subopt()**

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

#include <ViennaRNA/subopt.h>

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**

**15.17.3.4 subopt_circ()**

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

#include <ViennaRNA/subopt.h>

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**

**Returns**

## 15.18 Structure sampling from the ensemble

### 15.18.1 Detailed Description

Collaboration diagram for Structure sampling from 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 ∗ vrna_pbacktrack5 (vrna_fold_compound_t ∗vc, int length)

    *Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*

- char ∗ vrna_pbacktrack (vrna_fold_compound_t ∗vc)

    *Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.*

- 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.*

### 15.18.2 Function Documentation

#### 15.18.2.1 vrna_pbacktrack5()

```
char * vrna_pbacktrack5 (
            vrna_fold_compound_t * vc,
            int length )
```

`#include <ViennaRNA/boltzmann_sampling.h>`

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

**Precondition**

> Unique multiloop decomposition has to be active upon creation of `vc` with vrna_fold_compound() or similar. This can be done easily by passing vrna_fold_compound() a model details parameter with vrna_md_t.uniq_ML = 1.
> vrna_pf() has to be called first to fill the partition function matrices

---

**Parameters**

| *vc* | The fold compound data structure |
|---|---|
| *length* | The length of the subsequence to consider (starting with 5' end) |

**Returns**

A sampled secondary structure in dot-bracket notation (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method **pbacktrack()** to objects of type *fold_↩ compound*

**15.18.2.2 vrna_pbacktrack()**

```
char * vrna_pbacktrack (
            vrna_fold_compound_t * vc )
```

`#include <ViennaRNA/boltzmann_sampling.h>`

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

**Precondition**

Unique multiloop decomposition has to be active upon creation of vc with vrna_fold_compound() or similar. This can be done easily by passing vrna_fold_compound() a model details parameter with vrna_md_t.uniq_ML = 1.
vrna_pf() has to be called first to fill the partition function matrices

**Note**

This function is polymorphic. It accepts vrna_fold_compound_t of type VRNA_FC_TYPE_SINGLE, and VRNA_FC_TYPE_COMPARATIVE.
The function will automagically detect cicular RNAs based on the model_details in exp_params as provided via the vrna_fold_compound_t

**Parameters**

| *vc* | The fold compound data structure |
|---|---|

**Returns**

A sampled secondary structure in dot-bracket notation (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method **pbacktrack()** to objects of type *fold_↩ compound* that accepts an optional length argument. Hence, it serves as a replacement for vrna_pbacktrack().

**15.18.2.3 pbacktrack()**

```
char* pbacktrack (
            char * sequence )
```

#include <ViennaRNA/part_func.h>

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

**Precondition**

> st_back has to be set to 1 before calling pf_fold() or pf_fold_par()
> 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

**15.18.2.4 pbacktrack_circ()**

```
char* pbacktrack_circ (
            char * sequence )
```

#include <ViennaRNA/part_func.h>

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**

> st_back has to be set to 1 before calling pf_fold() or pf_fold_par()
> pf_fold_par() or pf_circ_fold() have to be called first to fill the partition function matrices

**Deprecated** Use vrna_pbacktrack() instead.

**Parameters**

| | |
|---|---|
| *sequence* | The RNA sequence |

**Returns**

> A sampled secondary structure in dot-bracket notation

## 15.18.3 Variable Documentation

### 15.18.3.1 st_back

```
int st_back
```

```
#include <ViennaRNA/part_func.h>
```

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

**Deprecated** set the *uniq_ML* flag in vrna_md_t before passing it to vrna_fold_compound().

**See also**

pbacktrack(), pbacktrack_circ

## 15.19 Compute the structure with maximum expected accuracy (MEA)

### 15.19.1 Detailed Description

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

**Functions**

- float MEA (plist *p, char *structure, double gamma)

  *Computes a MEA (maximum expected accuracy) structure.*

### 15.19.2 Function Documentation

#### 15.19.2.1 MEA()

```
float MEA (
            plist * p,
            char * structure,
            double gamma )
```

`#include <ViennaRNA/MEA.h>`

Computes a MEA (maximum expected accuracy) structure.

The algorithm maximizes the expected accuracy

$$A(S) = \sum_{(i,j)\in S} 2\gamma p_{ij} + \sum_{i\notin S} p_i^u$$

Higher values of $\gamma$ result in more base pairs of lower probability and thus higher sensitivity. Low values of $\gamma$ 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.

## 15.20 Compute the centroid structure

### 15.20.1 Detailed Description

Collaboration diagram for Compute the centroid structure:

**Functions**

- char * vrna_centroid (vrna_fold_compound_t *vc, double *dist)

    *Get the centroid structure of the ensemble.*
- char * vrna_centroid_from_plist (int length, double *dist, vrna_ep_t *pl)

    *Get the centroid structure of the ensemble.*
- char * vrna_centroid_from_probs (int length, double *dist, FLT_OR_DBL *probs)

    *Get the centroid structure of the ensemble.*

### 15.20.2 Function Documentation

#### 15.20.2.1 vrna_centroid()

```
char* vrna_centroid (
            vrna_fold_compound_t * vc,
            double * dist )
```

```
#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

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 $p_i j > 0.5$ The distance of the centroid to the ensemble is written to the memory adressed by *dist*.

**Parameters**

| in | *vc* | The fold compound data structure |
|---|---|---|
| out | *dist* | A pointer to the distance variable where the centroid distance will be written to |

**Returns**

The centroid structure of the ensemble in dot-bracket notation

**15.20.2.2   vrna_centroid_from_plist()**

```
char* vrna_centroid_from_plist (
            int length,
            double * dist,
            vrna_ep_t * pl )
```

`#include <ViennaRNA/centroid.h>`

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for centroid() with a vrna_ep_t 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**

| in | *length* | The length of the sequence |
|---|---|---|
| out | *dist* | A pointer to the distance variable where the centroid distance will be written to |
| in | *pl* | A pair list containing base pair probability information about the ensemble |

**Returns**

The centroid structure of the ensemble in dot-bracket notation

**15.20.2.3   vrna_centroid_from_probs()**

```
char* vrna_centroid_from_probs (
            int length,
            double * dist,
            FLT_OR_DBL * probs )
```

`#include <ViennaRNA/centroid.h>`

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**

| in | *length* | The length of the sequence |
|---|---|---|
| out | *dist* | A pointer to the distance variable where the centroid distance will be written to |
| in | *probs* | An upper triangular matrix containing base pair probabilities (access via iindx vrna_idx_row_wise() ) |

**Returns**

The centroid structure of the ensemble in dot-bracket notation

## 15.21 RNA-RNA interaction

### 15.21.1 Detailed Description

Collaboration diagram for RNA-RNA interaction:

**Modules**

- MFE Structures of two hybridized Sequences

    *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.*

- Partition Function for two hybridized Sequences

    *Partition Function Cofolding.*

- Partition Function for two hybridized Sequences as a stepwise Process

    *RNA-RNA interaction as a stepwise process.*

**Files**

- file cofold.h

    *MFE implementations for RNA-RNA interaction.*

- file concentrations.h

    *Concentration computations for RNA-RNA interactions.*

- file duplex.h

    *Functions for simple RNA-RNA duplex interactions.*

- file part_func_co.h

    *Partition function for two RNA sequences.*

- file part_func_up.h

    *Implementations for accessibility and RNA-RNA interaction as a stepwise process.*

## 15.22 Locally stable structures

### 15.22.1 Detailed Description

Collaboration diagram for Locally stable structures:

**Modules**

- Local MFE Structure Prediction and Z-scores
- Partition Function and Equilibrium Probabilities for Locally Stable Secondary Structures
- Local MFE Consensus Structure Prediction for Sequence Alignments

**Files**

- file Lfold.h

  *Functions for locally optimal MFE structure prediction.*

- file LPfold.h

  *Partition function implementation for the Lfold algorithm.*

## 15.23 Comparative structure prediction

### 15.23.1 Detailed Description

Collaboration diagram for Comparative structure prediction:

### 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 Structure Prediction for Sequence Alignments

### Files

- file alifold.h

  *Functions for comparative structure prediction using RNA sequence alignments.*

### Functions

- 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.*

- int get_alipf_arrays (short ∗∗∗S_p, short ∗∗∗S5_p, short ∗∗∗S3_p, unsigned short ∗∗∗a2s_p, char ∗∗∗Ss↩
  _p, FLT_OR_DBL ∗∗qb_p, FLT_OR_DBL ∗∗qm_p, FLT_OR_DBL ∗∗q1k_p, FLT_OR_DBL ∗∗qln_p, short
  ∗∗pscore)

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

- void update_alifold_params (void)

  *Update the energy parameters for alifold function.*

- float ∗∗ get_ribosum (const char ∗∗Alseq, int n_seq, int length)

  *Retrieve a RiboSum Scoring Matrix for a given Alignment.*

### 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.*

### 15.23.2 Function Documentation

**15.23.2.1  energy_of_alistruct()**

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

`#include <`ViennaRNA/alifold.h`>`

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

**Deprecated**  Usage of this function is discouraged!  Use vrna_eval_structure(), and vrna_eval_covar_structure() instead!

**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

**15.23.2.2  get_alipf_arrays()**

```
int get_alipf_arrays (
            short *** S_p,
            short *** S5_p,
            short *** S3_p,
            unsigned short *** a2s_p,
            char *** Ss_p,
            FLT_OR_DBL ** qb_p,
            FLT_OR_DBL ** qm_p,
            FLT_OR_DBL ** q1k_p,
            FLT_OR_DBL ** qln_p,
            short ** pscore )
```

`#include <`ViennaRNA/alifold.h`>`

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()

**Deprecated** It is discouraged to use this function! The new vrna_fold_compound_t allows direct access to all necessary consensus structure prediction related variables!

**See also**

vrna_fold_compound_t, vrna_fold_compound_comparative(), vrna_pf()

**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 alignment-column to sequence position mapping array |
| *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 ( $qln(l) = Q(l, n)$ ) |
| *pscore* | A pointer to the start of a pscore list |

**Returns**

Non Zero if everything went fine, 0 otherwise

**15.23.2.3   update_alifold_params()**

```
void update_alifold_params (
            void  )
```

#include <ViennaRNA/alifold.h>

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

**Deprecated** Usage of this function is discouraged! The new API uses vrna_fold_compound_t to lump all folding related necessities together, including the energy parameters. Use vrna_update_fold_params() to update the energy parameters within a vrna_fold_compound_t.

**15.23.3   Variable Documentation**

**15.23.3.1 cv_fact**

```
double cv_fact
```

```
#include <ViennaRNA/alifold.h>
```

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

**Deprecated** See vrna_md_t.cv_fact, and vrna_mfe() to avoid using global variables

Default is 1.

**15.23.3.2 nc_fact**

```
double nc_fact
```

```
#include <ViennaRNA/alifold.h>
```

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

**Deprecated** See vrna_md_t.nc_fact, and vrna_mfe() to avoid using global variables

Default is 1.

## 15.24 MFE Consensus Structures for Sequence Alignment(s)

### 15.24.1 Detailed Description

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

### Functions

- float vrna_alifold (const char ∗∗sequences, char ∗structure)

  *Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.*
- float vrna_circalifold (const char ∗∗sequences, char ∗structure)

  *Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.*
- 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.*

### 15.24.2 Function Documentation

#### 15.24.2.1 vrna_alifold()

```
float vrna_alifold (
            const char ** sequences,
            char * structure )
```

`#include <ViennaRNA/alifold.h>`

Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.

This simplified interface to vrna_mfe() computes the MFE and, if required, a consensus secondary structure for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_circalifold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

**Parameters**

| | |
|---|---|
| *sequences* | RNA sequence alignment |
| *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

**15.24.2.2   vrna_circalifold()**

```
float vrna_circalifold (
            const char ** sequences,
            char * structure )
```

`#include <`ViennaRNA/alifold.h`>`

Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.

This simplified interface to vrna_mfe() computes the MFE and, if required, a consensus secondary structure for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_alifold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

**Parameters**

| | |
|---|---|
| *sequences* | Sequence alignment of circular RNAs |
| *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

**15.24.2.3    alifold()**

```
float alifold (
            const char ** strings,
            char * structure )
```

#include <ViennaRNA/alifold.h>

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().

**Deprecated**  Usage of this function is discouraged! Use vrna_alifold(), or vrna_mfe() instead!

**See also**

>   vrna_alifold(), vrna_mfe()

**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

**15.24.2.4    circalifold()**

```
float circalifold (
            const char ** strings,
            char * structure )
```

#include <ViennaRNA/alifold.h>

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

**Deprecated**  Usage of this function is discouraged! Use vrna_alicircfold(), and vrna_mfe() instead!

**See also**

>   vrna_alicircfold(), vrna_alifold(), vrna_mfe()

**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

**15.24.2.5   free_alifold_arrays()**

```
void free_alifold_arrays (
            void  )
```

`#include <ViennaRNA/alifold.h>`

Free the memory occupied by MFE alifold functions.

**Deprecated**   Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced vrna_fold_compound_t is handled by vrna_fold_compound_free()

**See also**

vrna_fold_compound_free()

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

### 15.25.1 Detailed Description

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

**Functions**

- float vrna_pf_alifold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)

  *Compute Partition function $Q$ (and base pair probabilities) for an RNA sequence alignment using a comparative method.*
- float vrna_pf_circalifold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)

  *Compute Partition function $Q$ (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.*
- float alipf_fold_par (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl, vrna_exp_param_t ∗parameters, int calculate_bppm, int is_constrained, int is_circular)
- float alipf_fold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗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 vrna_pinfo_t structs. The list is terminated by the first entry with pi.i = 0.*
- float alipf_circ_fold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)
- FLT_OR_DBL ∗ export_ali_bppm (void)

  *Get a pointer to the base pair probability array.*
- void free_alipf_arrays (void)

  *Free the memory occupied by folding matrices allocated by alipf_fold, alipf_circ_fold, etc.*

### 15.25.2 Function Documentation

#### 15.25.2.1 vrna_pf_alifold()

```
float vrna_pf_alifold (
            const char ** sequences,
            char * structure,
            vrna_ep_t ** pl )
```

```
#include <ViennaRNA/alifold.h>
```

Compute Partition function $Q$ (and base pair probabilities) for an RNA sequence alignment using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_pf_circalifold(), vrna_pf(), vrna_fold_compound_comparative(), vrna_fold_compound_t

**Parameters**

| | |
|---|---|
| *sequences* | RNA sequence alignment |
| *structure* | A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL) |
| *pl* | A pointer to a list of vrna_ep_t to store pairing probabilities (Maybe NULL) |

**Returns**

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

**15.25.2.2   vrna_pf_circalifold()**

```
float vrna_pf_circalifold (
            const char ** sequences,
            char * structure,
            vrna_ep_t ** pl )
```

```
#include <ViennaRNA/alifold.h>
```

Compute Partition function $Q$ (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.

This simplified interface to vrna_pf() computes the partition function and, if required, base pair probabilities for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf(), and the data structure vrna_fold_compound_t instead.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

**See also**

vrna_pf_alifold(), vrna_pf(), vrna_fold_compound_comparative(), vrna_fold_compound_t

**Parameters**

| | |
|---|---|
| *sequences* | Sequence alignment of circular RNAs |
| *structure* | A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL) |
| *pl* | A pointer to a list of vrna_ep_t to store pairing probabilities (Maybe NULL) |

**Returns**

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

**15.25.2.3 alipf_fold_par()**

```
float alipf_fold_par (
            const char ** sequences,
            char * structure,
            vrna_ep_t ** pl,
            vrna_exp_param_t * parameters,
            int calculate_bppm,
            int is_constrained,
            int is_circular )
```

#include <ViennaRNA/alifold.h>

**Deprecated** Use vrna_pf() instead

**Parameters**

| | |
|---|---|
| *sequences* | |
| *structure* | |
| *pl* | |
| *parameters* | |
| *calculate_bppm* | |
| *is_constrained* | |
| *is_circular* | |

**Returns**

**15.25.2.4 alipf_fold()**

```
float alipf_fold (
            const char ** sequences,
            char * structure,
            vrna_ep_t ** pl )
```

#include <ViennaRNA/alifold.h>

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 vrna_pinfo_t structs. The list is terminated by the first entry with pi.i = 0.

**Deprecated** Use vrna_pf() instead

**Parameters**

| sequences | |
|---|---|
| structure | |
| pl | |

**Returns**

**15.25.2.5 alipf_circ_fold()**

```
float alipf_circ_fold (
            const char ** sequences,
            char * structure,
            vrna_ep_t ** pl )
```

#include <ViennaRNA/alifold.h>

**Deprecated** Use vrna_pf() instead

**Parameters**

| sequences | |
|---|---|
| structure | |
| pl | |

**Returns**

**15.25.2.6 export_ali_bppm()**

```
FLT_OR_DBL* export_ali_bppm (
            void  )
```

#include <ViennaRNA/alifold.h>

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];
```

**Deprecated** Usage of this function is discouraged! The new vrna_fold_compound_t allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to vrna_pf(), or any of the old API calls for consensus structure partition function folding.

**See also**

vrna_fold_compound_t, vrna_fold_compound_comparative(), and vrna_pf()

**Returns**

A pointer to the base pair probability array

**15.25.2.7   free_alipf_arrays()**

```
void free_alipf_arrays (
            void  )
```

#include <ViennaRNA/alifold.h>

Free the memory occupied by folding matrices allocated by alipf_fold, alipf_circ_fold, etc.

**Deprecated**  Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with vrna_) will be not affected!

**See also**

vrna_fold_compound_t, vrna_vrna_fold_compound_free()

## 15.26 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)

### 15.26.1 Detailed Description

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.*

### 15.26.2 Function Documentation

#### 15.26.2.1 alipbacktrack()

```
char* alipbacktrack (
            double * prob )
```

#include <ViennaRNA/alifold.h>

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

**Deprecated** Use vrna_pbacktrack() instead!

**Parameters**

| | |
|---|---|
| *prob* | to be described (berni) |

**Returns**

A sampled consensus secondary structure in dot-bracket notation

## 15.27 Classified Dynamic Programming variants

### 15.27.1 Detailed Description

Collaboration diagram for Classified Dynamic Programming variants:

**Modules**

- Distance based partitioning of the Secondary Structure Space
- Compute the Density of States

## 15.28 Distance based partitioning of the Secondary Structure Space

### 15.28.1 Detailed Description

Collaboration diagram for Distance based partitioning of the Secondary Structure Space:

### Modules

- Calculating MFE representatives of a Distance Based Partitioning

    *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.*

- Calculate Partition Functions of a Distance Based Partitioning

    *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.*

- Stochastic Backtracking of Structures from Distance Based Partitioning

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

### Files

- file 2Dfold.h

    *MFE structures for base pair distance classes.*

- file 2Dpfold.h

    *Partition function implementations for base pair distance classes.*

## 15.29 Calculating MFE representatives of a Distance Based Partitioning

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.

### 15.29.1 Detailed Description

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.

**See also**

> For further details, we refer to Lorenz et al. 2009 [11]

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

**Data Structures**

- struct vrna_sol_TwoD_t

    *Solution element returned from vrna_mfe_TwoD() More...*
- struct TwoDfold_vars

    *Variables compound for 2Dfold MFE folding. More...*

**Typedefs**

- typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t

    *Solution element returned from vrna_mfe_TwoD()*
- typedef struct TwoDfold_vars TwoDfold_vars

    *Variables compound for 2Dfold MFE folding.*

**Functions**

- vrna_sol_TwoD_t ∗ vrna_mfe_TwoD (vrna_fold_compound_t ∗vc, int distance1, int distance2)

    *Compute MFE's and representative for distance partitioning.*
- char ∗ vrna_backtrack5_TwoD (vrna_fold_compound_t ∗vc, int k, int l, unsigned int j)

    *Backtrack a minimum free energy structure from a 5' section of specified length.*
- 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.*
- vrna_sol_TwoD_t ∗ 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.*

## 15.29.2 Data Structure Documentation

### 15.29.2.1 struct vrna_sol_TwoD_t

Solution element returned from vrna_mfe_TwoD()

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**

vrna_mfe_TwoD()

**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.*

### 15.29.2.2 struct TwoDfold_vars

Variables compound for 2Dfold MFE folding.

**Deprecated** This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

Collaboration diagram for TwoDfold_vars:

**Data Fields**

- vrna_param_t ∗ 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].*

## 15.29.3 Typedef Documentation

### 15.29.3.1 vrna_sol_TwoD_t

```
typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t
```

```
#include <ViennaRNA/2Dfold.h>
```

Solution element returned from vrna_mfe_TwoD()

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**

vrna_mfe_TwoD()

**15.29.3.2 TwoDfold_vars**

typedef struct TwoDfold_vars TwoDfold_vars

#include <ViennaRNA/2Dfold.h>

Variables compound for 2Dfold MFE folding.

**Deprecated** This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

## 15.29.4 Function Documentation

**15.29.4.1 vrna_mfe_TwoD()**

vrna_sol_TwoD_t* vrna_mfe_TwoD (
            vrna_fold_compound_t * vc,
            int distance1,
            int distance2 )

#include <ViennaRNA/2Dfold.h>

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**

vrna_fold_compound_TwoD(), vrna_fold_compound_free(), vrna_pf_TwoD() vrna_backtrack5_TwoD(), vrna_sol_TwoD_t, vrna_fold_compound_t

**Parameters**

| vc | The datastructure containing all precomputed folding attributes |
|---|---|
| distance1 | maximum distance to reference1 (-1 means no restriction) |
| distance2 | maximum distance to reference2 (-1 means no restriction) |

**Returns**

A list of minimum free energies (and corresponding structures) for each distance class

**15.29.4.2 vrna_backtrack5_TwoD()**

```
char* vrna_backtrack5_TwoD (
            vrna_fold_compound_t * vc,
            int k,
            int l,
            unsigned int j )
```

#include <ViennaRNA/2Dfold.h>

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

This function allows one 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 vrna_mfe_TwoD() belong to.

**Note**

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

**See also**

vrna_mfe_TwoD()

**Parameters**

| vc | The datastructure containing all precomputed folding attributes |
|---|---|
| j | The length in nucleotides beginning from the 5' end |
| k | distance to reference1 (may be -1) |
| l | distance to reference2 |

**15.29.4.3 get_TwoDfold_variables()**

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

#include <ViennaRNA/2Dfold.h>

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

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↵ compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**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

### 15.29.4.4 destroy_TwoDfold_variables()

```
void destroy_TwoDfold_variables (
            TwoDfold_vars * our_variables )
```

```
#include <ViennaRNA/2Dfold.h>
```

Destroy a TwoDfold_vars datastructure without memory loss.

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

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↵ compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**Parameters**

| our_variables | A pointer to the datastructure to be destroyed |
|---|---|

### 15.29.4.5 TwoDfoldList()

```
vrna_sol_TwoD_t* TwoDfoldList (
            TwoDfold_vars * vars,
            int distance1,
            int distance2 )
```

#include <ViennaRNA/2Dfold.h>

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.

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_mfe_TwoD(), and vrna_fold_compound_free() instead!

**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) |

### 15.29.4.6 TwoDfold_backtrack_f5()

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

#include <ViennaRNA/2Dfold.h>

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

This function allows one 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 TwoDfold() belong to.

**Note**

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

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_mfe_TwoD(), vrna_backtrack5_TwoD(), and vrna_fold_compound_free() instead!

**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 |

# 15.30 Calculate Partition Functions of a Distance Based Partitioning

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.

## 15.30.1 Detailed Description

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.

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

### Data Structures

- struct vrna_sol_TwoD_pf_t

    *Solution element returned from vrna_pf_TwoD() More...*

### Typedefs

- typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t

    *Solution element returned from vrna_pf_TwoD()*

### Functions

- vrna_sol_TwoD_pf_t ∗ vrna_pf_TwoD (vrna_fold_compound_t ∗vc, int maxDistance1, int maxDistance2)

    *Compute the partition function for all distance classes.*

## 15.30.2 Data Structure Documentation

### 15.30.2.1 struct vrna_sol_TwoD_pf_t

Solution element returned from vrna_pf_TwoD()

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**

vrna_pf_TwoD()

**Data Fields**

- int k

    *Distance to first reference.*
- int l

    *Distance to second reference.*
- FLT_OR_DBL q

    *partition function*

## 15.30.3 Typedef Documentation

### 15.30.3.1 vrna_sol_TwoD_pf_t

typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t

#include <ViennaRNA/2Dpfold.h>

Solution element returned from vrna_pf_TwoD()

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**

vrna_pf_TwoD()

## 15.30.4 Function Documentation

### 15.30.4.1 vrna_pf_TwoD()

vrna_sol_TwoD_pf_t* vrna_pf_TwoD (
            vrna_fold_compound_t * vc,
            int maxDistance1,
            int maxDistance2 )

#include <ViennaRNA/2Dpfold.h>

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 vrna_mfe_TwoD() 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 value of INF in the attribute 'k' of the returned list denotes the end of the list

**See also**

vrna_fold_compound_TwoD(), vrna_fold_compound_free(), vrna_fold_compound vrna_sol_TwoD_pf_t

**Parameters**

| | |
|---|---|
| *vc* | 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 corresponding distance classes

## 15.31 Stochastic Backtracking of Structures from Distance Based Partitioning

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

### 15.31.1 Detailed Description

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

Collaboration diagram for Stochastic Backtracking of Structures from Distance Based Partitioning:

**Functions**

- char ∗ vrna_pbacktrack_TwoD (vrna_fold_compound_t ∗vc, int d1, int d2)

    *Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- char ∗ vrna_pbacktrack5_TwoD (vrna_fold_compound_t ∗vc, 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.*

### 15.31.2 Function Documentation

#### 15.31.2.1 vrna_pbacktrack_TwoD()

```
char* vrna_pbacktrack_TwoD (
            vrna_fold_compound_t * vc,
            int d1,
            int d2 )
```

```
#include <ViennaRNA/2Dpfold.h>
```

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 vrna_pf_TwoD() preceding this function is mandatory!

**See also**

> vrna_pf_TwoD()

**Parameters**

| in,out | vc | The vrna_fold_compound_t datastructure containing all necessary folding attributes and matrices |
|--------|-----|-------------------------------------------------------------------------------------------------|
| in | d1 | The distance to reference1 (may be -1) |
| in | d2 | The distance to reference2 |

**Returns**

A sampled secondary structure in dot-bracket notation

**15.31.2.2 vrna_pbacktrack5_TwoD()**

```
char* vrna_pbacktrack5_TwoD (
            vrna_fold_compound_t * vc,
            int d1,
            int d2,
            unsigned int length )
```

#include <ViennaRNA/2Dpfold.h>

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 vrna_pbacktrack_TwoD() 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 vrna_pf_TwoD() preceding this function is mandatory!

**See also**

vrna_pbacktrack_TwoD(), vrna_pf_TwoD()

**Parameters**

| in,out | vc | The vrna_fold_compound_t datastructure containing all necessary folding attributes and matrices |
|--------|--------|-------------------------------------------------------------------------------------------------|
| in | d1 | The distance to reference1 (may be -1) |
| in | d2 | The distance to reference2 |
| in | length | The length of the structure beginning from the 5' end |

**Returns**

A sampled secondary structure in dot-bracket notation

## 15.32 Compute the Density of States

### 15.32.1 Detailed Description

Collaboration diagram for Compute the Density of States:

**Variables**

- int density_of_states [MAXDOS+1]

    *The Density of States.*

### 15.32.2 Variable Documentation

#### 15.32.2.1 density_of_states

```
int density_of_states[MAXDOS+1]
```

```
#include <ViennaRNA/subopt.h>
```

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()

## 15.33 Inverse Folding (Design)

RNA sequence design.

### 15.33.1 Detailed Description

RNA sequence design.

### 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

### 15.33.2 Function Documentation

#### 15.33.2.1 inverse_fold()

```
float inverse_fold (
          char * start,
          const char * target )
```

`#include <ViennaRNA/inverse.h>`

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

**15.33.2.2   inverse_pf_fold()**

```
float inverse_pf_fold (
            char * start,
            const char * target )
```

`#include <`ViennaRNA/inverse.h`>`

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

**15.33.3   Variable Documentation**

**15.33.3.1   final_cost**

```
float final_cost
```

`#include <`ViennaRNA/inverse.h`>`

when to stop inverse_pf_fold()

**15.33.3.2 give_up**

```
int give_up
```

`#include <`ViennaRNA/inverse.h`>`

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

**15.33.3.3 inv_verbose**

```
int inv_verbose
```

`#include <`ViennaRNA/inverse.h`>`

print out substructure on which inverse_fold() fails

## 15.34   Neighborhood relation and move sets for secondary structures

Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.

### 15.34.1   Detailed Description

Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.

This module contains methods to compute the neighbors of an RNA secondary structure. Neighbors of a given structure are all structures that differ in exactly one base pair. That means one can insert an delete base pairs in the given structure. These insertions and deletions of base pairs are usually called moves. A third move which is considered in these methods is a shift move. A shifted base pair has one stable position and one position that changes. These moves are encoded as follows:

- insertion: (i, j) where i,j $> 0$

- deletion: (i, j) where i,j $< 0$
  shift: (i, j) where either i $> 0$, j $< 0$ or i $< 0$, j $> 0$
  The negative position of a shift indicates the position that has changed.

```
Example:
      We have given a sequence and a structure.
      Sequence  AAGGAAACC
      Structure ..(.....)
      Indices   123456789

      The given base pair is (3,9) and the neighbors are the insertion (4, 8), the deletion (-3,-9), the
  shift (3,-8)
      and the shift (-4, 9).
      This leads to the neighbored structures:
      ...(....)
      .........
      ...(...).
      ....(...)
```
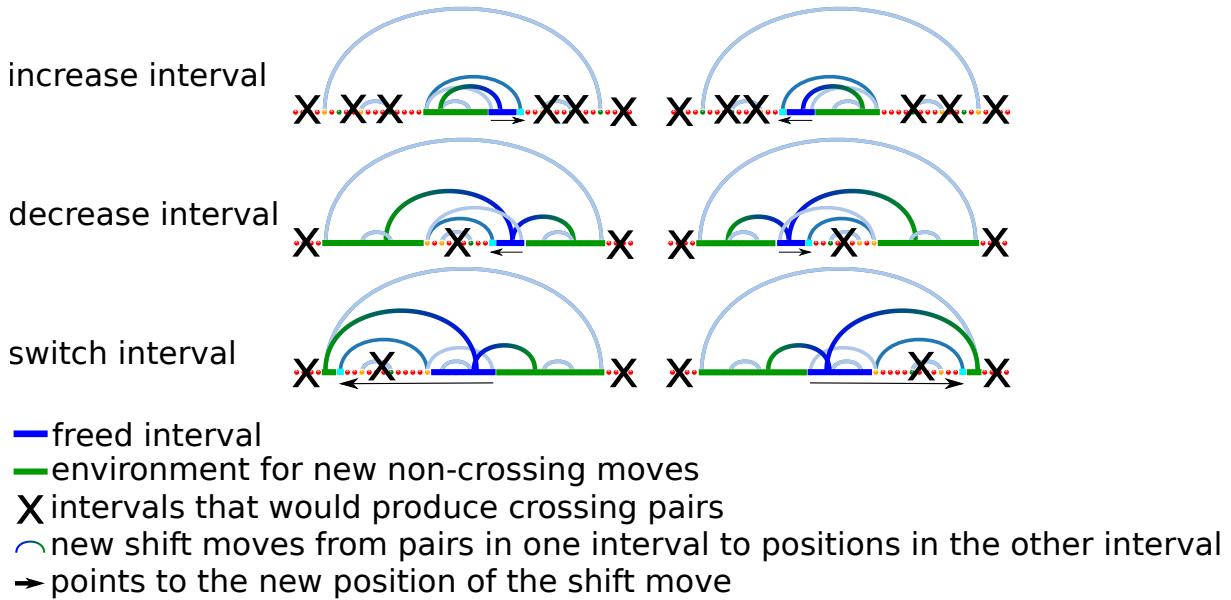
A simple method to construct all insertions is to iterate over the positions of a sequence twice. The first iteration has the index i in [1, sequence length], the second iteration has the index j in [i+1, sequence length]. All pairs (i,j) with compatible letters and which are non-crossing with present base pairs are valid neighbored insertion moves. Valid deletion moves are all present base pairs with negative sign. Valid shift moves are constructed by taking all paired positions as fix position of a shift move and iterating over all positions of the sequence. If the letters of a position are compatible and if it the move is non-crossing with existing base pairs, we have a valid shift move. The method of generating shift moves can be accelerated by skipping neighbored base pairs.

If we need to construct all neighbors several times for subsequent moves, we can speed up the task by using the move set of the previous structure. The previous move set has to be filtered, such that all moves that would cross the next selected move are non-crossing. Next, the selected move has to be removed. Then one has to only to generate all moves that were not possible before. One move is the inverted selected move (if it was an insertion, simply make the indices negative). The generation of all other new moves is different and depends on the selected move. It is easy for an insertion move, because we have only to include all non-crossing shift moves, that are possible with the new base pair. For that we can either iterate over the sequence or we can select all crossing shift moves in the filter procedure and convert them into shifts.

The generation of new moves given a deletion is a little bit more complex, because we can create more moves. At first we can insert the deleted pair as insertion move. Then we generate all insertions that would have crossed the deleted base pair. Finally we construct all crossing shift moves.

If the given move is a shift, we can save much time by specifying the intervals for the generation of new moves. The interval which was enclosed by the positive position of the shift move and the previous paired position is the freed interval after applying the move. This freed interval includes all positions and base pairs that we need to construct new insertions and shifts. All these new moves have one position in the freed interval and the other position in the environment of the freed interval. The environment are all position which are outside the freed interval, but within the same enclosing loop of the shift move. The environment for valid base pairs can be divided into one or more intervals, depending on the shift move. The following examples describe a few scenarios to specify the intervals of the environment.



Given the intervals of the environment and the freed interval, the new shift moves can be constructed quickly. One has to take all positions of pairs from the environment in order to create valid pairs with positions in the freed interval. The same procedure can be applied for the other direction. This is taking all paired positions within the freed interval in order to look for pairs with valid positions in the intervals of the environment. Collaboration diagram for Neighborhood relation and move sets for secondary structures:

## Files

- file neighbor.h

  *Methods to compute the neighbors of an RNA secondary structure.*

## Data Structures

- struct vrna_move_s

  *An atomic representation of the transition / move from one structure to its neighbor. More...*

**Macros**

- #define VRNA_MOVESET_INSERTION 4

    *Option flag indicating insertion move.*
- #define VRNA_MOVESET_DELETION 8

    *Option flag indicating deletion move.*
- #define VRNA_MOVESET_SHIFT 16

    *Option flag indicating shift move.*
- #define VRNA_MOVESET_NO_LP 32

    *Option flag indicating moves without lonely base pairs.*
- #define VRNA_MOVESET_DEFAULT (VRNA_MOVESET_INSERTION | VRNA_MOVESET_DELETION)

    *Option flag indicating default move set, i.e. insertions/deletion of a base pair.*

**Functions**

- void vrna_move_list_free (vrna_move_t ∗moves)
- void vrna_move_apply (short ∗pt, const vrna_move_t ∗m)

    *Apply a particular move / transition to a secondary structure, i.e. transform a structure.*
- void vrna_loopidx_update (int ∗loopidx, const short ∗pt, int length, const vrna_move_t ∗m)

    *Alters the loopIndices array that was constructed with vrna_loopidx_from_ptable().*
- vrna_move_t ∗ vrna_neighbors (vrna_fold_compound_t ∗vc, const short ∗pt, unsigned int options)

    *Generate neighbors of a secondary structure.*
- vrna_move_t ∗ vrna_neighbors_successive (const vrna_fold_compound_t ∗vc, const vrna_move_t ∗curr↩
    _move, const short ∗prev_pt, const vrna_move_t ∗prev_neighbors, int size_prev_neighbors, int ∗size_↩
    neighbors, unsigned int options)

    *Generate neighbors of a secondary structure (the fast way)*

## 15.34.2 Data Structure Documentation

### 15.34.2.1 struct vrna_move_s

An atomic representation of the transition / move from one structure to its neighbor.

An atomic transition / move may be (a) the insertion of a base pair (both fields are positive), (b) the deletion of a base pair (both fields are negative), or (c) a base pair shift where one position stays constant while the other is allowed to shift along the same loop it resides in (one field position and the other negative, where the positive field indicates the constant position and the absolute value of the negative field is the new position of the pairing partner).

A value of 0 is either field is typically used to indicate the lists last element.

Collaboration diagram for vrna_move_s:

**Data Fields**

- int pos_5
- int pos_3
- vrna_move_t ∗ next

**15.34.2.1.1  Field Documentation**

**15.34.2.1.1.1  pos_5**

`int vrna_move_s::pos_5`

The 5' position of a base pair, or any position of a shifted pair

**15.34.2.1.1.2  pos_3**

`int vrna_move_s::pos_3`

The 3' position of a base pair, or any position of a shifted pair

**15.34.2.1.1.3  next**

[vrna_move_t](#)* vrna_move_s::next

The next base pair (if an elementary move changes more than one base pair) Has to be terminated with move 0,0

**15.34.3  Macro Definition Documentation**

**15.34.3.1  VRNA_MOVESET_INSERTION**

`#define VRNA_MOVESET_INSERTION 4`

`#include <`[ViennaRNA/neighbor.h](#)`>`

Option flag indicating insertion move.

**See also**

  [vrna_neighbors()](#), [vrna_neighbors_successive](#), [vrna_path()](#)

**15.34.3.2  VRNA_MOVESET_DELETION**

`#define VRNA_MOVESET_DELETION 8`

`#include <`[ViennaRNA/neighbor.h](#)`>`

Option flag indicating deletion move.

**See also**

  [vrna_neighbors()](#), [vrna_neighbors_successive](#), [vrna_path()](#)

**15.34.3.3 VRNA_MOVESET_SHIFT**

```
#define VRNA_MOVESET_SHIFT 16
```

#include <ViennaRNA/neighbor.h>

Option flag indicating shift move.

**See also**

vrna_neighbors(), vrna_neighbors_successive, vrna_path()

**15.34.3.4 VRNA_MOVESET_NO_LP**

```
#define VRNA_MOVESET_NO_LP 32
```

#include <ViennaRNA/neighbor.h>

Option flag indicating moves without lonely base pairs.

**See also**

vrna_neighbors(), vrna_neighbors_successive, vrna_path()

**15.34.3.5 VRNA_MOVESET_DEFAULT**

```
#define VRNA_MOVESET_DEFAULT (VRNA_MOVESET_INSERTION | VRNA_MOVESET_DELETION)
```

#include <ViennaRNA/neighbor.h>

Option flag indicating default move set, i.e. insertions/deletion of a base pair.

**See also**

vrna_neighbors(), vrna_neighbors_successive, vrna_path()

**15.34.4 Function Documentation**

**15.34.4.1 vrna_move_list_free()**

```
void vrna_move_list_free (
            vrna_move_t * moves )
```

#include <ViennaRNA/neighbor.h>

delete all moves in a zero terminated list.

**15.34.4.2 vrna_move_apply()**

```
void vrna_move_apply (
            short * pt,
            const vrna_move_t * m )
```

#include <ViennaRNA/neighbor.h>

Apply a particular move / transition to a secondary structure, i.e. transform a structure.

**Parameters**

| in,out | *pt* | The pair table representation of the secondary structure |
|--------|------|----------------------------------------------------------|
| in     | *m*  | The move to apply                                        |

**15.34.4.3 vrna_loopidx_update()**

```
void vrna_loopidx_update (
            int * loopidx,
            const short * pt,
            int length,
            const vrna_move_t * m )
```

`#include <ViennaRNA/neighbor.h>`

Alters the loopIndices array that was constructed with vrna_loopidx_from_ptable().

The loopIndex of the current move will be inserted. The correctness of the input will not be checked because the speed should be optimized.

**Parameters**

| in,out | *loopidx* | The loop index data structure that needs an update |
|--------|-----------|---------------------------------------------------|
| in     | *pt*      | A pair table on which the move will be executed    |
|        | *length*  | The length of the structure                        |
| in     | *m*       | The move that is applied to the current structure  |

**15.34.4.4 vrna_neighbors()**

```
vrna_neighbors (
            vrna_fold_compound_t * vc,
            const short * pt,
            unsigned int options )
```

`#include <ViennaRNA/neighbor.h>`

Generate neighbors of a secondary structure.

This function allows one to generate all structural neighbors (according to a particular move set) of an RNA secondary structure. The neighborhood is then returned as a list of transitions / moves required to transform the current structure into the actual neighbor.

**See also**

> vrna_neighbors_successive(), vrna_move_apply(), VRNA_MOVESET_INSERTION, VRNA_MOVESET_DELETION, VRNA_MOVESET_SHIFT, VRNA_MOVESET_DEFAULT

**Parameters**

| in | *vc* | A vrna_fold_compound_t containing the energy parameters and model details |
|----|------|-------------------------------------------------------------------------|
| in | *pt* | The pair table representation of the structure |
|    | *options* | Options to modify the behavior of this function, e.g. available move set |

**Returns**

Neighbors as a list of moves / transitions (the last element in the list has both of its fields set to 0)

**SWIG Wrapper Notes** This function is attached as an overloaded method *neighbors()* to objects of type *fold_↩ compound*. The optional parameter `options` defaults to VRNA_MOVESET_DEFAULT if it is omitted.

**15.34.4.5 vrna_neighbors_successive()**

```
vrna_move_t* vrna_neighbors_successive (
            const vrna_fold_compound_t * vc,
            const vrna_move_t * curr_move,
            const short * prev_pt,
            const vrna_move_t * prev_neighbors,
            int size_prev_neighbors,
            int * size_neighbors,
            unsigned int options )
```

#include <ViennaRNA/neighbor.h>

Generate neighbors of a secondary structure (the fast way)

This function implements a fast way to generate all neighbors of a secondary structure that results from successive applications of individual moves. The speed-up results from updating an already known list of valid neighbors before the individual move towards the current structure took place. In essence, this function removes neighbors that are not accessible anymore and inserts neighbors emerging after a move took place.

**See also**

vrna_neighbors(), vrna_move_apply(), VRNA_MOVESET_INSERTION, VRNA_MOVESET_DELETION, VRNA_MOVESET_SHIFT, VRNA_MOVESET_DEFAULT

**Parameters**

| in | *vc* | A vrna_fold_compound_t containing the energy parameters and model details |
|----|------|-------------------------------------------------------------------------|
| in | *curr_move* | The move that was/will be applied to `prev_pt` |
| in | *prev_pt* | A pair table representation of the structure before `curr_move` is/was applied |
| in | *prev_neighbors* | The list of neighbors of `prev_pt` |
|    | *size_prev_neighbors* | The size of `prev_neighbors`, i.e. the lists length |
| out | *size_neighbors* | A pointer to store the size / length of the new neighbor list |
|    | *options* | Options to modify the behavior of this function, e.g. available move set |

**Returns**

Neighbors as a list of moves / transitions (the last element in the list has both of its fields set to 0)

## 15.35 Refolding paths of secondary structures

### 15.35.1 Detailed Description

Collaboration diagram for Refolding paths of secondary structures:

### Modules

- Direct refolding paths between two secondary structures

    *Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.*

### Files

- file findpath.h

    *A breadth-first search heuristic for optimal direct folding paths.*
- file walk.h

    *Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.*

### Macros

- #define VRNA_PATH_STEEPEST_DESCENT 128

    *Option flag to request a steepest descent / gradient path.*
- #define VRNA_PATH_RANDOM 256

    *Option flag to request a random walk path.*
- #define VRNA_PATH_NO_TRANSITION_OUTPUT 512

    *Option flag to omit returning the transition path.*
- #define VRNA_PATH_DEFAULT (VRNA_PATH_STEEPEST_DESCENT | VRNA_MOVESET_DEFAULT)

    *Option flag to request defaults (steepest descent / default move set)*

### Functions

- vrna_move_t ∗ vrna_path (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int steps, unsigned int options)

    *Compute a path, store the final structure, and return a list of transition moves from the start to the final structure.*
- vrna_move_t ∗ vrna_path_gradient (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int options)

    *Compute a steepest descent / gradient path, store the final structure, and return a list of transition moves from the start to the final structure.*
- vrna_move_t ∗ vrna_path_random (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int steps, unsigned int options)

    *Generate a random walk / path of a given length, store the final structure, and return a list of transition moves from the start to the final structure.*

### 15.35.2 Macro Definition Documentation

**15.35.2.1 VRNA_PATH_STEEPEST_DESCENT**

`#define VRNA_PATH_STEEPEST_DESCENT 128`

`#include <`[ViennaRNA/walk.h](#)`>`

Option flag to request a steepest descent / gradient path.

**See also**

> [vrna_path()](#)

**15.35.2.2 VRNA_PATH_RANDOM**

`#define VRNA_PATH_RANDOM 256`

`#include <`[ViennaRNA/walk.h](#)`>`

Option flag to request a random walk path.

**See also**

> [vrna_path()](#)

**15.35.2.3 VRNA_PATH_NO_TRANSITION_OUTPUT**

`#define VRNA_PATH_NO_TRANSITION_OUTPUT 512`

`#include <`[ViennaRNA/walk.h](#)`>`

Option flag to omit returning the transition path.

**See also**

> [vrna_path()](#), [vrna_path_gradient()](#), [vrna_path_random()](#)

**15.35.2.4 VRNA_PATH_DEFAULT**

`#define VRNA_PATH_DEFAULT (`[VRNA_PATH_STEEPEST_DESCENT](#) `|` [VRNA_MOVESET_DEFAULT](#)`)`

`#include <`[ViennaRNA/walk.h](#)`>`

Option flag to request defaults (steepest descent / default move set)

**See also**

> [vrna_path()](#), [VRNA_PATH_STEEPEST_DESCENT](#), [VRNA_MOVESET_DEFAULT](#)

### 15.35.3 Function Documentation

#### 15.35.3.1 vrna_path()

```
vrna_path (
            vrna_fold_compound_t * vc,
            short * pt,
            unsigned int steps,
            unsigned int options )
```

`#include <ViennaRNA/walk.h>`

Compute a path, store the final structure, and return a list of transition moves from the start to the final structure.

This function computes, given a start structure in pair table format, a transition path, updates the pair table to the final structure of the path. Finally, if not requested otherwise by using the VRNA_PATH_NO_TRANSITION_OUTPUT flag in the `options` field, this function returns a list of individual transitions that lead from the start to the final structure if requested.

The currently available transition paths are

- Steepest Descent / Gradient walk (flag: VRNA_PATH_STEEPEST_DESCENT)

- Random walk (flag: VRNA_PATH_RANDOM)

The type of transitions must be set through the `options` parameter

**Note**

Since the result is written to the input structure you may want to use vrna_ptable_copy() before calling this function to keep the initial structure

**See also**

vrna_path_gradient(), vrna_path_random(), vrna_ptable(), vrna_ptable_copy(), vrna_fold_compound() VRNA_PATH_STEEPEST_DESCENT, VRNA_PATH_RANDOM, VRNA_MOVESET_DEFAULT, VRNA_MOVESET_SHIFT, VRNA_PATH_NO_TRANSITION_OUTPUT

**Parameters**

| in | vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|---|
| in,out | pt | The pair table containing the start structure. Used to update to the final structure after execution of this function |
| in | options | Options to modify the behavior of this function |

**Returns**

A list of transition moves (default), or NULL (if options & VRNA_PATH_NO_TRANSITION_OUTPUT)

This function is attached as an overloaded method *path()* to objects of type *fold_compound*. The optional parameter `options` defaults to VRNA_PATH_DEFAULT if it is omitted.

**15.35.3.2 vrna_path_gradient()**

```
vrna_path_gradient (
            vrna_fold_compound_t * vc,
            short * pt,
            unsigned int options )
```

`#include <ViennaRNA/walk.h>`

Compute a steepest descent / gradient path, store the final structure, and return a list of transition moves from the start to the final structure.

This function computes, given a start structure in pair table format, a steepest descent path, updates the pair table to the final structure of the path. Finally, if not requested otherwise by using the VRNA_PATH_NO_TRANSITION_OUTPUT flag in the `options` field, this function returns a list of individual transitions that lead from the start to the final structure if requested.

**Note**

Since the result is written to the input structure you may want to use vrna_ptable_copy() before calling this function to keep the initial structure

**See also**

vrna_path_random(), vrna_path(), vrna_ptable(), vrna_ptable_copy(), vrna_fold_compound() VRNA_MOVESET_DEFAULT, VRNA_MOVESET_SHIFT, VRNA_PATH_NO_TRANSITION_OUTPUT

**Parameters**

| in | vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|---|
| in,out | pt | The pair table containing the start structure. Used to update to the final structure after execution of this function |
| in | options | Options to modify the behavior of this function |

**Returns**

A list of transition moves (default), or NULL (if options & VRNA_PATH_NO_TRANSITION_OUTPUT)

**SWIG Wrapper Notes** This function is attached as an overloaded method *path_gradient()* to objects of type *fold↩_compound*. The optional parameter `options` defaults to VRNA_PATH_DEFAULT if it is omitted.

**15.35.3.3   vrna_path_random()**

```
vrna_path_random (
            vrna_fold_compound_t * vc,
            short * pt,
            unsigned int steps,
            unsigned int options )
```

`#include <ViennaRNA/walk.h>`

Generate a random walk / path of a given length, store the final structure, and return a list of transition moves from the start to the final structure.

This function generates, given a start structure in pair table format, a random walk / path, updates the pair table to the final structure of the path. Finally, if not requested otherwise by using the VRNA_PATH_NO_TRANSITION_OUTPUT flag in the `options` field, this function returns a list of individual transitions that lead from the start to the final structure if requested.

**Note**

> Since the result is written to the input structure you may want to use vrna_ptable_copy() before calling this function to keep the initial structure

**See also**

> vrna_path_gradient(), vrna_path(), vrna_ptable(), vrna_ptable_copy(), vrna_fold_compound() VRNA_MOVESET_DEFAULT, VRNA_MOVESET_SHIFT, VRNA_PATH_NO_TRANSITION_OUTPUT

**Parameters**

| in | vc | A vrna_fold_compound_t containing the energy parameters and model details |
|---|---|---|
| in,out | pt | The pair table containing the start structure. Used to update to the final structure after execution of this function |
| in | steps | The length of the path, i.e. the total number of transitions / moves |
| in | options | Options to modify the behavior of this function |

**Returns**

> A list of transition moves (default), or NULL (if options & VRNA_PATH_NO_TRANSITION_OUTPUT)

**SWIG Wrapper Notes**   This function is attached as an overloaded method *path_gradient()* to objects of type *fold↩ _compound*. The optional parameter `options` defaults to VRNA_PATH_DEFAULT if it is omitted.

## 15.36 Experimental structure probing data

Include experimental structure probing data to guide structure predictions.

### 15.36.1 Detailed Description

Include experimental structure probing data to guide structure predictions.

Collaboration diagram for Experimental structure probing data:

**Modules**

- SHAPE reactivity data

    *Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.*
- Generate soft constraints from data

    *Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.*

## 15.37 SHAPE reactivity data

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

### 15.37.1 Detailed Description

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

Details for our implementation to incorporate SHAPE reactivity data to guide secondary structure prediction can be found in [13] Collaboration diagram for SHAPE reactivity data:

**Files**

- file constraints_SHAPE.h

    *This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.*

**Functions**

- int vrna_sc_add_SHAPE_deigan (vrna_fold_compound_t ∗vc, const double ∗reactivities, double m, double b, unsigned int options)

    *Add SHAPE reactivity data as soft constraints (Deigan et al. method)*

- int vrna_sc_add_SHAPE_deigan_ali (vrna_fold_compound_t ∗vc, const char ∗∗shape_files, const int ∗shape_file_association, double m, double b, unsigned int options)

    *Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)*

- int vrna_sc_add_SHAPE_zarringhalam (vrna_fold_compound_t ∗vc, const double ∗reactivities, double b, double default_value, const char ∗shape_conversion, unsigned int options)

    *Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)*

- int vrna_sc_SHAPE_to_pr (const char ∗shape_conversion, double ∗values, int length, double default_value)

    *Convert SHAPE reactivity values to probabilities for being unpaired.*

### 15.37.2 Function Documentation

**15.37.2.1 vrna_sc_add_SHAPE_deigan()**

```
int vrna_sc_add_SHAPE_deigan (
            vrna_fold_compound_t * vc,
            const double * reactivities,
            double m,
            double b,
            unsigned int options )
```

#include <ViennaRNA/constraints_SHAPE.h>

Add SHAPE reactivity data as soft constraints (Deigan et al. method)

This approach of SHAPE directed RNA folding uses the simple linear ansatz

$$\Delta G_{\mathsf{SHAPE}}(i) = m \ln(\mathsf{SHAPE\ reactivity}(i) + 1) + b$$

to convert SHAPE reactivity values to pseudo energies whenever a nucleotide $i$ contributes to a stacked pair. A positive slope $m$ penalizes high reactivities in paired regions, while a negative intercept $b$ results in a confirmatory "bonus" free energy for correctly predicted base pairs. Since the energy evaluation of a base pair stack involves two pairs, the pseudo energies are added for all four contributing nucleotides. Consequently, the energy term is applied twice for pairs inside a helix and only once for pairs adjacent to other structures. For all other loop types the energy model remains unchanged even when the experimental data highly disagrees with a certain motif.

**See also**

For further details, we refer to [4].
vrna_sc_remove(), vrna_sc_add_SHAPE_zarringhalam(), vrna_sc_minimize_pertubation()

**Parameters**

| | |
|---|---|
| vc | The vrna_fold_compound_t the soft constraints are associated with |
| reactivities | A vector of normalized SHAPE reactivities |
| m | The slope of the conversion function |
| b | The intercept of the conversion function |
| options | The options flag indicating how/where to store the soft constraints |

**Returns**

1 on successful extraction of the method, 0 on errors

**SWIG Wrapper Notes** This function is attached as method **sc_add_SHAPE_deigan()** to objects of type *fold_↩ compound*

**15.37.2.2 vrna_sc_add_SHAPE_deigan_ali()**

```
int vrna_sc_add_SHAPE_deigan_ali (
            vrna_fold_compound_t * vc,
            const char ** shape_files,
```

```
                    const int * shape_file_association,
                    double m,
                    double b,
                    unsigned int options )
```

#include <ViennaRNA/constraints_SHAPE.h>

Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)

**Parameters**

| vc | The vrna_fold_compound_t the soft constraints are associated with |
|---|---|
| shape_files | A set of filenames that contain normalized SHAPE reactivity data |
| shape_file_association | An array of integers that associate the files with sequences in the alignment |
| m | The slope of the conversion function |
| b | The intercept of the conversion function |
| options | The options flag indicating how/where to store the soft constraints |

**Returns**

  1 on successful extraction of the method, 0 on errors

**SWIG Wrapper Notes**  This function is attached as method **sc_add_SHAPE_deigan_ali()** to objects of type *fold*↩ *_compound*

**15.37.2.3   vrna_sc_add_SHAPE_zarringhalam()**

```
int vrna_sc_add_SHAPE_zarringhalam (
            vrna_fold_compound_t * vc,
            const double * reactivities,
            double b,
            double default_value,
            const char * shape_conversion,
            unsigned int options )
```

#include <ViennaRNA/constraints_SHAPE.h>

Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)

This method first converts the observed SHAPE reactivity of nucleotide $i$ into a probability $q_i$ that position $i$ is unpaired by means of a non-linear map. Then pseudo-energies of the form

$$\Delta G_{\text{SHAPE}}(x, i) = \beta \, |x_i - q_i|$$

are computed, where $x_i = 0$ if position $i$ is unpaired and $x_i = 1$ if $i$ is paired in a given secondary structure. The parameter $\beta$ serves as scaling factor. The magnitude of discrepancy between prediction and experimental observation is represented by $|x_i - q_i|$.

**See also**

  For further details, we refer to [22]
  vrna_sc_remove(), vrna_sc_add_SHAPE_deigan(), vrna_sc_minimize_pertubation()

**Parameters**

| vc | The vrna_fold_compound_t the soft constraints are associated with |
|---|---|
| reactivities | A vector of normalized SHAPE reactivities |
| b | The scaling factor $\beta$ of the conversion function |
| default_value | The default value for a nucleotide where reactivity data is missing for |
| shape_conversion | A flag that specifies how to convert reactivities to probabilities |
| options | The options flag indicating how/where to store the soft constraints |

**Returns**

1 on successful extraction of the method, 0 on errors

**SWIG Wrapper Notes**  This function is attached as method **sc_add_SHAPE_zarringhalam()** to objects of type *fold_compound*

**15.37.2.4  vrna_sc_SHAPE_to_pr()**

```
int vrna_sc_SHAPE_to_pr (
            const char * shape_conversion,
            double * values,
            int length,
            double default_value )
```

#include <ViennaRNA/constraints_SHAPE.h>

Convert SHAPE reactivity values to probabilities for being unpaired.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the FLT_OR_DBL array values.

**See also**

vrna_file_SHAPE_read()

**Parameters**

| shape_conversion | String definining the method used for the conversion process |
|---|---|
| values | Pointer to an array of SHAPE reactivities |
| length | Length of the array of SHAPE reactivities |
| default_value | Result used for position with invalid/missing reactivity values |

## 15.38 Generate soft constraints from data

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

### 15.38.1 Detailed Description

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Collaboration diagram for Generate soft constraints from data:

### Files

- file perturbation_fold.h

    *Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.*

### Macros

- #define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

    *Use the sum of squared aberrations as objective function.*
- #define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

    *Use the sum of absolute aberrations as objective function.*
- #define VRNA_MINIMIZER_DEFAULT 0

    *Use a custom implementation of the gradient descent algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_CONJUGATE_FR 1

    *Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_CONJUGATE_PR 2

    *Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_VECTOR_BFGS 3

    *Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_VECTOR_BFGS2 4

    *Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_STEEPEST_DESCENT 5

    *Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.*

### Typedefs

- typedef void(∗ progress_callback) (int iteration, double score, double ∗epsilon)

    *Callback for following the progress of the minimization process.*

**Functions**

- void [vrna_sc_minimize_pertubation](#) ([vrna_fold_compound_t](#) *vc, const double *q_prob_unpaired, int objective_function, double sigma_squared, double tau_squared, int algorithm, int sample_size, double *epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, [progress_callback](#) callback)

    *Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.*

## 15.38.2 Macro Definition Documentation

### 15.38.2.1 VRNA_OBJECTIVE_FUNCTION_QUADRATIC

`#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0`

`#include <`[ViennaRNA/perturbation_fold.h](#)`>`

Use the sum of squared aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^{n} \frac{\epsilon_i^2}{\tau^2} + \sum_{i=1}^{n} \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \to min$$

### 15.38.2.2 VRNA_OBJECTIVE_FUNCTION_ABSOLUTE

`#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1`

`#include <`[ViennaRNA/perturbation_fold.h](#)`>`

Use the sum of absolute aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^{n} \frac{|\epsilon_i|}{\tau^2} + \sum_{i=1}^{n} \frac{|p_i(\vec{\epsilon}) - q_i|}{\sigma^2} \to min$$

### 15.38.2.3 VRNA_MINIMIZER_CONJUGATE_FR

`#define VRNA_MINIMIZER_CONJUGATE_FR 1`

`#include <`[ViennaRNA/perturbation_fold.h](#)`>`

Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

### 15.38.2.4 VRNA_MINIMIZER_CONJUGATE_PR

`#define VRNA_MINIMIZER_CONJUGATE_PR 2`

`#include <`[ViennaRNA/perturbation_fold.h](#)`>`

Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

**15.38.2.5 VRNA_MINIMIZER_VECTOR_BFGS**

```
#define VRNA_MINIMIZER_VECTOR_BFGS 3
```

`#include <ViennaRNA/perturbation_fold.h>`

Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

**15.38.2.6 VRNA_MINIMIZER_VECTOR_BFGS2**

```
#define VRNA_MINIMIZER_VECTOR_BFGS2 4
```

`#include <ViennaRNA/perturbation_fold.h>`

Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

**15.38.2.7 VRNA_MINIMIZER_STEEPEST_DESCENT**

```
#define VRNA_MINIMIZER_STEEPEST_DESCENT 5
```

`#include <ViennaRNA/perturbation_fold.h>`

Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

**15.38.3 Typedef Documentation**

**15.38.3.1 progress_callback**

```
typedef void(* progress_callback) (int iteration, double score, double *epsilon)
```

`#include <ViennaRNA/perturbation_fold.h>`

Callback for following the progress of the minimization process.

**Parameters**

| | |
|---|---|
| *iteration* | The number of the current iteration |
| *score* | The score of the objective function |
| *epsilon* | The perturbation vector yielding the reported score |

## 15.38.4 Function Documentation

### 15.38.4.1 vrna_sc_minimize_pertubation()

```
void vrna_sc_minimize_pertubation (
            vrna_fold_compound_t * vc,
            const double * q_prob_unpaired,
            int objective_function,
            double sigma_squared,
            double tau_squared,
            int algorithm,
            int sample_size,
            double * epsilon,
            double initialStepSize,
            double minStepSize,
            double minImprovement,
            double minimizerTolerance,
            progress_callback callback )
```

`#include <ViennaRNA/perturbation_fold.h>`

Find a vector of perturbation energies that minimizes the discripancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Use an iterative minimization algorithm to find a vector of perturbation energies whose incorporation as soft constraints shifts the predicted pairing probabilities closer to the experimentally observed probabilities. The algorithm aims to minimize an objective function that penalizes discrepancies between predicted and observed pairing probabilities and energy model adjustments, i.e. an appropriate vector of perturbation energies satisfies

$$F(\vec{\epsilon}) = \sum_{\mu} \frac{\epsilon_{\mu}^2}{\tau^2} + \sum_{i=1}^{n} \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \to \min.$$

An initialized fold compound and an array containing the observed probability for each nucleotide to be unbound are required as input data. The parameters objective_function, sigma_squared and tau_squared are responsible for adjusting the aim of the objective function. Dependend on which type of objective function is selected, either squared or absolute aberrations are contributing to the objective function. The ratio of the parameters sigma_↩squared and tau_squared can be used to adjust the algorithm to find a solution either close to the thermodynamic prediction (sigma_squared $>>$ tau_squared) or close to the experimental data (tau_squared $>>$ sigma_squared). The minimization can be performed by makeing use of a custom gradient descent implementation or using one of the minimizing algorithms provided by the GNU Scientific Library. All algorithms require the evaluation of the gradient of the objective function, which includes the evaluation of conditional pairing probabilites. Since an exact evaluation is expensive, the probabilities can also be estimated from sampling by setting an appropriate sample size. The found vector of perturbation energies will be stored in the array epsilon. The progress of the minimization process can be tracked by implementing and passing a callback function.

**See also**

For further details we refere to [20].

**Parameters**

| vc | Pointer to a fold compound |
|---|---|

**Parameters**

| | |
|---|---|
| *q_prob_unpaired* | Pointer to an array containing the probability to be unpaired for each nucleotide |
| *objective_function* | The type of objective function to be used (VRNA_OBJECTIVE_FUNCTION_QUADRATIC / VRNA_OBJECTIVE_FUNCTION_LINEAR) |
| *sigma_squared* | A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the null vector. |
| *tau_squared* | A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the data provided in q_prob_unpaired. |
| *algorithm* | The minimization algorithm (VRNA_MINIMIZER_$*$) |
| *sample_size* | The number of sampled sequences used for estimating the pairing probabilities. A value $<= 0$ will lead to an exact evaluation. |
| *epsilon* | A pointer to an array used for storing the calculated vector of perturbation energies |
| *callback* | A pointer to a callback function used for reporting the current minimization progress |

## 15.39 Ligands binding to RNA structures

Simple extensions to model ligand binding to RNA structures.

### 15.39.1 Detailed Description

Simple extensions to model ligand binding to RNA structures.

Collaboration diagram for Ligands binding to RNA structures:

**Modules**

- Ligands binding to unstructured domains

    *Add ligand binding to loop regions using the Unstructured domains feature.*
- Incorporating ligands binding to specific sequence/structure motifs using soft constraints

    *Ligand binding to specific hairpin/interior loop like motifs using the Soft constraints feature.*

**Files**

- file constraints_ligand.h

    *Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.*

## 15.40 Ligands binding to unstructured domains

Add ligand binding to loop regions using the Unstructured domains feature.

Sometime, certain ligands, like single strand binding (SSB) proteins, compete with intramolecular base pairing of the RNA. In situations, where the dissociation constant of the ligand is known and the ligand binds to a consecutive stretch of single-stranded nucleotides we can use the Unstructured domains functionality to extend the RNA folding grammar. This module provides a convenience default implementation that covers most of the application scenarios.

The function vrna_ud_add_motif() attaches a ligands sequence motif and corresponding binding free energy to the list of known ligand motifs within a vrna_fold_compound_t.domains_up attribute. The first call to this function initializes the Unstructured domains feature with our default implementation. Subsequent calls of secondary structure predciction algorithms with the modified vrna_fold_compound_t then directly include the competition of the ligand with regules base pairing. Since we utilize the unstructured domain extension, The ligand binding model can be removed again using the vrna_ud_remove() function. Collaboration diagram for Ligands binding to unstructured domains:
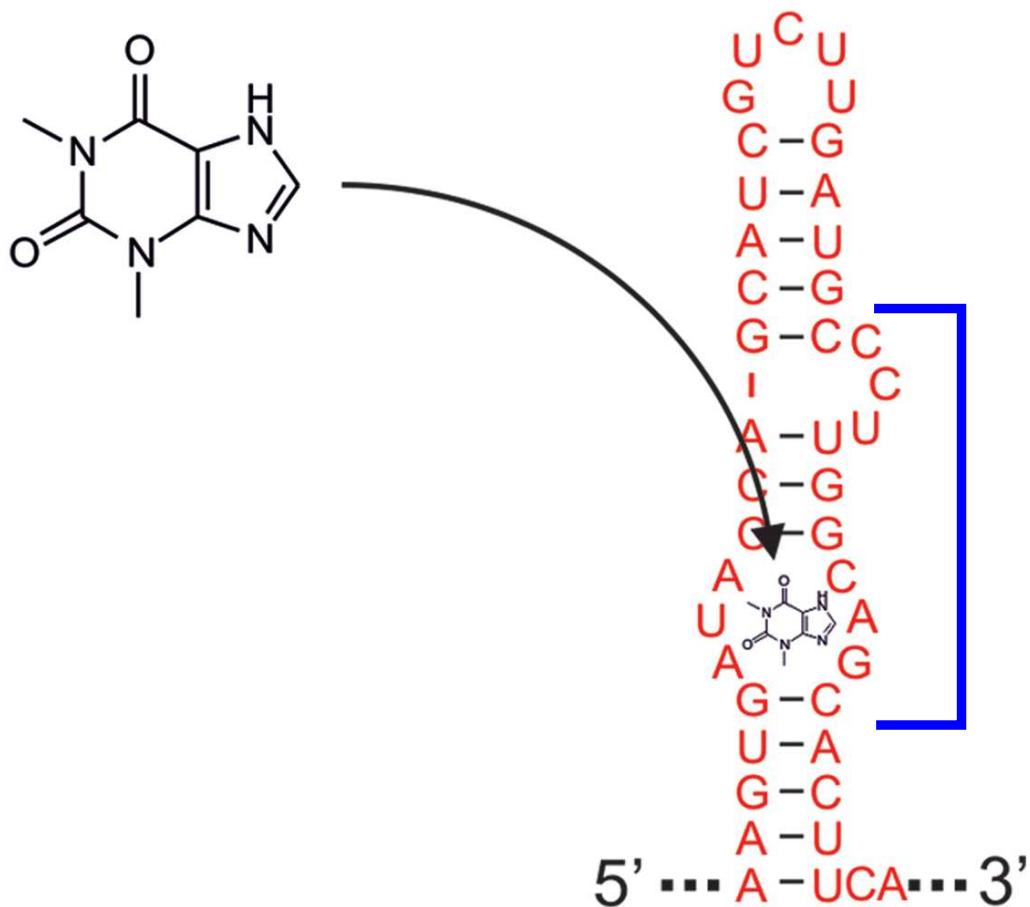
## 15.41 Incorporating ligands binding to specific sequence/structure motifs using soft constraints

Ligand binding to specific hairpin/interior loop like motifs using the Soft constraints feature.

### 15.41.1 Detailed Description

Ligand binding to specific hairpin/interior loop like motifs using the Soft constraints feature.

Here is an example that adds a theophylline binding motif. Free energy contribution is derived from $k_d = 0.32 \mu mol/l$, taken from Jenison et al. 1994



```
vrna_sc_add_hi_motif(vc,
                     "GAUACCAG&CCCUUGGCAGC",
                     "(...(((((&)...)))...)",
                     -9.22, VRNA_OPTION_DEFAULT);
```

Collaboration diagram for Incorporating ligands binding to specific sequence/structure motifs using soft constraints:

## Functions

- int vrna_sc_add_hi_motif (vrna_fold_compound_t ∗vc, const char ∗seq, const char ∗structure, FLT_OR_DBL energy, unsigned int options)

    *Add soft constraints for hairpin or interior loop binding motif.*

### 15.41.2   Function Documentation

#### 15.41.2.1   vrna_sc_add_hi_motif()

```
int vrna_sc_add_hi_motif (
            vrna_fold_compound_t * vc,
            const char * seq,
            const char * structure,
            FLT_OR_DBL energy,
            unsigned int options )
```

#include <ViennaRNA/constraints_ligand.h>

Add soft constraints for hairpin or interior loop binding motif.

**Parameters**

| vc | The vrna_fold_compound_t the motif is applied to |
| --- | --- |
| seq | The sequence motif (may be interspaced by '&' character |
| structure | The structure motif (may be interspaced by '&' character |
| energy | The free energy of the motif (e.g. binding free energy) |
| options | Options |

**Returns**

non-zero value if application of the motif using soft constraints was successful

**SWIG Wrapper Notes**  This function is attached as method **sc_add_hi_motif()** to objects of type *fold_compound*

## 15.42 Complex structured modules

### 15.42.1 Detailed Description

Collaboration diagram for Complex structured modules:

**Modules**

- G-quadruplexes

  *Various functions related to G-quadruplex computations.*

**Files**

- file gquad.h

  *G-quadruplexes.*

## 15.43 G-quadruplexes

Various functions related to G-quadruplex computations.

### 15.43.1 Detailed Description

Various functions related to G-quadruplex computations.

Collaboration diagram for G-quadruplexes:

**Functions**

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

### 15.43.2 Function Documentation

#### 15.43.2.1 get_gquad_matrix()

```
int* get_gquad_matrix (
            short * S,
            vrna_param_t * P )
```

```
#include <ViennaRNA/gquad.h>
```

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

**15.43.2.2  parse_gquad()**

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

#include <ViennaRNA/gquad.h>

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; ... ;

**15.43.2.3  backtrack_GQuad_IntLoop()**

```
PRIVATE int backtrack_GQuad_IntLoop (
            int c,
            int i,
            int j,
            int type,
            short * S,
            int * ggg,
            int * index,
            int * p,
            int * q,
            vrna_param_t * P )
```

#include <ViennaRNA/gquad.h>

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

### 15.43.2.4  backtrack_GQuad_IntLoop_L()

```
PRIVATE int backtrack_GQuad_IntLoop_L (
            int c,
            int i,
            int j,
            int type,
            short * S,
            int ** ggg,
            int maxdist,
            int * p,
            int * q,
            vrna_param_t * P )
```

`#include <ViennaRNA/gquad.h>`

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

## 15.44 Data Structures and Preprocessor Macros

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

### 15.44.1 Detailed Description

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

Collaboration diagram for Data Structures and Preprocessor Macros:

**Modules**

- The Fold Compound

  *This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.*
- The Dynamic Programming Matrices

  *This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.*

**Files**

- file data_structures.h

  *Various data structures and pre-processor macros.*
- file dp_matrices.h

  *Functions to deal with standard dynamic programming (DP) matrices.*

**Data Structures**

- struct vrna_basepair_s

  *Base pair data structure used in subopt.c. More...*
- struct vrna_cpair_s

  *this datastructure is used as input parameter in functions of PS_dot.c More...*
- struct vrna_color_s
- struct vrna_data_linear_s
- struct vrna_sect_s

  *Stack of partial structures for backtracking. More...*
- struct vrna_bp_stack_s

  *Base pair stack element. More...*
- struct pu_contrib

  *contributions to p_u More...*
- struct interact

  *interaction data structure for RNAup More...*
- struct pu_out

  *Collection of all free_energy of beeing unpaired values for output. More...*
- struct constrain

  *constraints for cofolding More...*

- struct duplexT

  *Data structure for RNAduplex. More...*
- struct node

  *Data structure for RNAsnoop (fold energy list) More...*
- struct snoopT

  *Data structure for RNAsnoop. More...*
- struct dupVar

  *Data structure used in RNApkplex. More...*

## Typedefs

- typedef struct vrna_basepair_s vrna_basepair_t

  *Typename for the base pair repesenting data structure vrna_basepair_s.*
- typedef struct vrna_elem_prob_s vrna_plist_t

  *Typename for the base pair list repesenting data structure vrna_elem_prob_s.*
- typedef struct vrna_bp_stack_s vrna_bp_stack_t

  *Typename for the base pair stack repesenting data structure vrna_bp_stack_s.*
- typedef struct vrna_cpair_s vrna_cpair_t

  *Typename for data structure vrna_cpair_s.*
- typedef struct vrna_sect_s vrna_sect_t

  *Typename for stack of partial structures vrna_sect_s.*
- typedef double FLT_OR_DBL

  *Typename for floating point number in partition function computations.*
- typedef struct vrna_basepair_s PAIR

  *Old typename of vrna_basepair_s.*
- typedef struct vrna_elem_prob_s plist

  *Old typename of vrna_elem_prob_s.*
- typedef struct vrna_cpair_s cpair

  *Old typename of vrna_cpair_s.*
- typedef struct vrna_sect_s sect

  *Old typename of vrna_sect_s.*
- typedef struct vrna_bp_stack_s bondT

  *Old typename of vrna_bp_stack_s.*
- typedef struct pu_contrib pu_contrib

  *contributions to p_u*
- typedef struct interact interact

  *interaction data structure for RNAup*
- typedef struct pu_out pu_out

  *Collection of all free_energy of beeing unpaired values for output.*
- typedef struct constrain constrain

  *constraints for cofolding*
- typedef struct node folden

  *Data structure for RNAsnoop (fold energy list)*
- typedef struct dupVar dupVar

  *Data structure used in RNApkplex.*

## Functions

- void vrna_C11_features (void)

  *Dummy symbol to check whether the library was build using C11/C++11 features.*

## 15.44.2 Data Structure Documentation

### 15.44.2.1 struct vrna_basepair_s

Base pair data structure used in subopt.c.

### 15.44.2.2 struct vrna_cpair_s

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

### 15.44.2.3 struct vrna_color_s

### 15.44.2.4 struct vrna_data_linear_s

Collaboration diagram for vrna_data_linear_s:

### 15.44.2.5 struct vrna_sect_s

Stack of partial structures for backtracking.

### 15.44.2.6 struct vrna_bp_stack_s

Base pair stack element.

### 15.44.2.7 struct pu_contrib

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*

**15.44.2.8   struct interact**

interaction data structure for RNAup

**Data Fields**

- double ∗ Pi

  *probabilities of interaction*
- double ∗ Gi

  *free energies of interaction*
- double Gikjl

  *full free energy for interaction between [k,i] k<i in longer seq and [j,l] j<l in shorter seq*
- double Gikjl_wo

  *Gikjl without contributions for prob_unpaired.*
- int i

  *k<i in longer seq*
- int k

  *k<i in longer seq*
- int j

  *j<l in shorter seq*
- int l

  *j<l in shorter seq*
- int length

  *length of longer sequence*

**15.44.2.9   struct pu_out**

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*

**15.44.2.10   struct constrain**

constraints for cofolding

**15.44.2.11 struct duplexT**

Data structure for RNAduplex.

**15.44.2.12 struct node**

Data structure for RNAsnoop (fold energy list)

Collaboration diagram for node:

**15.44.2.13 struct snoopT**

Data structure for RNAsnoop.

**15.44.2.14 struct dupVar**

Data structure used in RNApkplex.

## 15.44.3 Typedef Documentation

**15.44.3.1 PAIR**

typedef struct vrna_basepair_s PAIR

#include <ViennaRNA/data_structures.h>

Old typename of vrna_basepair_s.

**Deprecated** Use vrna_basepair_t instead!

**15.44.3.2 plist**

typedef struct vrna_elem_prob_s plist

#include <ViennaRNA/data_structures.h>

Old typename of vrna_elem_prob_s.

**Deprecated** Use vrna_ep_t or vrna_elem_prob_s instead!

**15.44.3.3 cpair**

typedef struct vrna_cpair_s cpair

#include <ViennaRNA/data_structures.h>

Old typename of vrna_cpair_s.

**Deprecated** Use vrna_cpair_t instead!

**15.44.3.4 sect**

typedef struct vrna_sect_s sect

#include <ViennaRNA/data_structures.h>

Old typename of vrna_sect_s.

**Deprecated** Use vrna_sect_t instead!

**15.44.3.5 bondT**

typedef struct vrna_bp_stack_s bondT

#include <ViennaRNA/data_structures.h>

Old typename of vrna_bp_stack_s.

**Deprecated** Use vrna_bp_stack_t instead!

**15.44.4 Function Documentation**

**15.44.4.1  vrna_C11_features()**

```
void vrna_C11_features (
            void  )
```

```
#include <ViennaRNA/data_structures.h>
```

Dummy symbol to check whether the library was build using C11/C++11 features.

By default, several data structures of our new v3.0 API use C11/C++11 features, such as unnamed unions, unnamed structs. However, these features can be deactivated at compile time to allow building the library and executables with compilers that do not support these features.

Now, the problem arises that once our static library is compiled and a third-party application is supposed to link against it, it needs to know, at compile time, how to correctly address particular data structures. This is usually implicitely taken care of through the API exposed in our header files. Unfortunately, we had some preprocessor directives in our header files that changed the API depending on the capabilities of the compiler the third-party application is build with. This in turn prohibited the use of an RNAlib compiled without C11/C++11 support in a program that compiles/links with enabled C11/C++11 support and vice-versa.

Therefore, we introduce this dummy symbol which can be used to check, whether the static library was build with C11/C++11 features.

**Note**

> If the symbol is present, the library was build with enabled C11/C++11 features support and no action is required. However, if the symbol is missing in RNAlib $>=$ 2.2.9, programs that link to RNAlib must define a pre-processor identifier *VRNA_DISABLE_C11_FEATURES* before including any ViennaRNA Package header file, for instance by adding a *CPPFLAG*
>
> ```
> CPPFLAGS+=-DVRNA_DISABLE_C11_FEATURES
> ```

**Since**

> v2.2.9

## 15.45 Utilities

### 15.45.1 Detailed Description

Collaboration diagram for Utilities:

### Modules

- Parsing, converting, and comparing sequences
- Parsing, converting, comparing secondary structures
- Utilities for sequence alignments
- Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments

    *Functions dealing with file formats for RNA sequences, structures, and alignments.*
- Functions that provide dynamically buffered stream-like data structures
- Functions for Creating Secondary Structure Plots, Dot-Plots, and More
- Functions to convert between various units

### Files

- file aln_util.h

    *Various utility- and helper-functions for sequence alignments and comparative structure prediction.*
- file alphabet.h

    *Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.*
- file combinatorics.h

    *Various implementations that deal with combinatorial aspects of RNA/DNA folding.*
- file sequence.h

    *Functions and data structures related to sequence representations.*
- file string_utils.h

    *General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.*
- file structure_utils.h

    *Various utility- and helper-functions for secondary structure parsing, converting, etc.*
- file utils.h

    *General utility- and helper-functions used throughout the ViennaRNA Package.*

### Macros

- #define VRNA_INPUT_ERROR 1U

    *Output flag of get_input_line(): "An ERROR has occured, maybe EOF".*
- #define VRNA_INPUT_QUIT 2U

    *Output flag of get_input_line(): "the user requested quitting the program".*
- #define VRNA_INPUT_MISC 4U

    *Output flag of get_input_line(): "something was read".*
- #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.*
- #define VRNA_INPUT_CONSTRAINT 32U

  *Input flag for get_input_line():*

  *Tell get_input_line() that we assume to read a structure constraint.*

- #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".*

- #define VRNA_INPUT_NO_REST 512U

  *Input switch for vrna_file_fasta_read_record(): "do fill rest array".*

- #define VRNA_INPUT_NO_SPAN 1024U

  *Input switch for vrna_file_fasta_read_record(): "never allow data to span more than one line".*

- #define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U

  *Input switch for vrna_file_fasta_read_record(): "do not skip empty lines".*

- #define VRNA_INPUT_BLANK_LINE 4096U

  *Output flag for vrna_file_fasta_read_record(): "read an empty line".*

- #define VRNA_INPUT_NOSKIP_COMMENTS 128U

  *Input switch for get_input_line(): "do not skip comment lines".*

- #define VRNA_INPUT_COMMENT 8192U

  *Output flag for vrna_file_fasta_read_record(): "read a comment".*

- #define MIN2(A, B) ((A) < (B) ? (A) : (B))

  *Get the minimum of two comparable values.*

- #define MAX2(A, B) ((A) > (B) ? (A) : (B))

  *Get the maximum of two comparable values.*

- #define MIN3(A, B, C) (MIN2( (MIN2((A),(B))) ,(C)))

  *Get the minimum of three comparable values.*

- #define MAX3(A, B, C) (MAX2( (MAX2((A),(B))) ,(C)))

  *Get the maximum of three comparable values.*

## Functions

- char ∗ vrna_ptypes (const short ∗S, vrna_md_t ∗md)

  *Get an array of the numerical encoding for each possible base pair (i,j)*

- short ∗ vrna_seq_encode (const char ∗sequence, vrna_md_t ∗md)

  *Get a numerical representation of the nucleotide sequence.*

- short ∗ vrna_seq_encode_simple (const char ∗sequence, vrna_md_t ∗md)

  *Get a numerical representation of the nucleotide sequence (simple version)*

- int vrna_nucleotide_encode (char c, vrna_md_t ∗md)

  *Encode a nucleotide character to numerical value.*

- char vrna_nucleotide_decode (int enc, vrna_md_t ∗md)

  *Decode a numerical representation of a nucleotide back into nucleotide alphabet.*

- unsigned int ∗∗ vrna_enumerate_necklaces (const unsigned int ∗type_counts)

  *Enumerate all necklaces with fixed content.*

- void ∗ vrna_alloc (unsigned size)

  *Allocate space safely.*

- void ∗ vrna_realloc (void ∗p, unsigned size)

  *Reallocate space safely.*

- void vrna_message_error (const char ∗format,...)

  *Print an error message and die.*

- void vrna_message_verror (const char ∗format, va_list args)

  *Print an error message and die.*

- void vrna_message_warning (const char ∗format,...)

  *Print a warning message.*

- void vrna_message_vwarning (const char ∗format, va_list args)

*Print a warning message.*

- void vrna_message_info (FILE ∗fp, const char ∗format,...)

    *Print an info message.*

- void vrna_message_vinfo (FILE ∗fp, const char ∗format, va_list args)

    *Print an info message.*

- void vrna_init_rand (void)

    *Initialize seed for random number generator.*

- double vrna_urn (void)

    *get a random number from [0..1]*

- int vrna_int_urn (int from, int to)

    *Generates a pseudo random integer in a specified range.*

- char ∗ vrna_time_stamp (void)

    *Get a timestamp.*

- unsigned int get_input_line (char ∗∗string, unsigned int options)

- void vrna_message_input_seq_simple (void)

    *Print a line to stdout that asks for an input sequence.*

- void vrna_message_input_seq (const char ∗s)

    *Print a line with a user defined string and a ruler to stdout.*

- int ∗ vrna_idx_row_wise (unsigned int length)

    *Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.*

- int ∗ vrna_idx_col_wise (unsigned int length)

    *Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.*

## Variables

- unsigned short xsubi [3]

    *Current 48 bit random number.*

### 15.45.2 Macro Definition Documentation

#### 15.45.2.1 VRNA_INPUT_FASTA_HEADER

```
#define VRNA_INPUT_FASTA_HEADER 8U
```

```
#include <ViennaRNA/utils.h>
```

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

#### 15.45.2.2 VRNA_INPUT_CONSTRAINT

```
#define VRNA_INPUT_CONSTRAINT 32U
```

```
#include <ViennaRNA/utils.h>
```

Input flag for get_input_line():
Tell get_input_line() that we assume to read a structure constraint.

## 15.45.3 Function Documentation

### 15.45.3.1 vrna_ptypes()

```
char* vrna_ptypes (
            const short * S,
            vrna_md_t * md )
```

#include <ViennaRNA/alphabet.h>

Get an array of the numerical encoding for each possible base pair (i,j)

**Note**

> This array is always indexed in column-wise order, in contrast to previously different indexing between mfe and pf variants!

**See also**

> vrna_idx_col_wise(), vrna_fold_compound_t

### 15.45.3.2 vrna_nucleotide_encode()

```
int vrna_nucleotide_encode (
            char c,
            vrna_md_t * md )
```

#include <ViennaRNA/alphabet.h>

Encode a nucleotide character to numerical value.

This function encodes a nucleotide character to its numerical representation as required by many functions in R↩ NAlib.

**See also**

> vrna_nucleotide_decode(), vrna_seq_encode()

**Parameters**

| c | The nucleotide character to encode |
|---|---|
| md | The model details that determine the kind of encoding |

**Returns**

> The encoded nucleotide

**15.45.3.3 vrna_nucleotide_decode()**

```
char vrna_nucleotide_decode (
            int enc,
            vrna_md_t * md )
```

#include <ViennaRNA/alphabet.h>

Decode a numerical representation of a nucleotide back into nucleotide alphabet.

This function decodes a numerical representation of a nucleotide character back into nucleotide alphabet

**See also**

> vrna_nucleotide_encode(), vrna_seq_encode()

**Parameters**

| enc | The encoded nucleotide |
|-----|-------------------------|
| md  | The model details that determine the kind of decoding |

**Returns**

> The decoded nucleotide character

**15.45.3.4 vrna_enumerate_necklaces()**

```
unsigned int ** vrna_enumerate_necklaces (
            const unsigned int * type_counts )
```

#include <ViennaRNA/combinatorics.h>

Enumerate all necklaces with fixed content.

This function implements *A fast algorithm to generate necklaces with fixed content* as published by Joe Sawada in 2003 [16].

The function receives a list of counts (the elements on the necklace) for each type of object within a necklace. The list starts at index 0 and ends with an entry that has a count of 0. The algorithm then enumerates all non-cyclic permutations of the content, returned as a list of necklaces. This list, again, is zero-terminated, i.e. the last entry of the list is a NULL pointer.

**Parameters**

| *type_counts* | A 0-terminated list of entity counts |
|---|---|

**Returns**

A list of all non-cyclic permutations of the entities

**SWIG Wrapper Notes** This function is available as global function **enumerate_necklaces()** which accepts lists input, an produces list of lists output.

**15.45.3.5    vrna_alloc()**

```
void* vrna_alloc (
            unsigned size )
```

#include <ViennaRNA/utils.h>

Allocate space safely.

**Parameters**

| *size* | The size of the memory to be allocated in bytes |
|---|---|

**Returns**

A pointer to the allocated memory

**15.45.3.6    vrna_realloc()**

```
void* vrna_realloc (
            void * p,
            unsigned size )
```

#include <ViennaRNA/utils.h>

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

**15.45.3.7 vrna_message_error()**

```
void vrna_message_error (
            const char * format,
             ... )
```

#include <ViennaRNA/utils.h>

Print an error message and die.

This function is a wrapper to *fprintf*(stderr, ...) that puts a capital **ERROR:** in front of the message and then exits the calling program.

**See also**

> vrna_message_verror(), vrna_message_warning(), vrna_message_info()

**Parameters**

| format | The error message to be printed |
|--------|---------------------------------|
| ... | Optional arguments for the formatted message string |

**15.45.3.8 vrna_message_verror()**

```
void vrna_message_verror (
            const char * format,
            va_list args )
```

#include <ViennaRNA/utils.h>

Print an error message and die.

This function is a wrapper to *vfprintf*(stderr, ...) that puts a capital **ERROR:** in front of the message and then exits the calling program.

**See also**

> vrna_message_error(), vrna_message_warning(), vrna_message_info()

**Parameters**

| format | The error message to be printed |
|--------|---------------------------------|
| args | The argument list for the formatted message string |

**15.45.3.9 vrna_message_warning()**

```
void vrna_message_warning (
            const char * format,
             ... )
```

#include <ViennaRNA/utils.h>

Print a warning message.

This function is a wrapper to *fprintf*(stderr, ...) that puts a capital **WARNING:** in front of the message.

**See also**

vrna_message_vwarning(), vrna_message_error(), vrna_message_info()

**Parameters**

| format | The warning message to be printed |
|--------|-----------------------------------|
| *...* | Optional arguments for the formatted message string |

**15.45.3.10 vrna_message_vwarning()**

```
void vrna_message_vwarning (
            const char * format,
            va_list args )
```

#include <ViennaRNA/utils.h>

Print a warning message.

This function is a wrapper to *fprintf*(stderr, ...) that puts a capital **WARNING:** in front of the message.

**See also**

vrna_message_vwarning(), vrna_message_error(), vrna_message_info()

**Parameters**

| format | The warning message to be printed |
|--------|-----------------------------------|
| *args* | The argument list for the formatted message string |

**15.45.3.11 vrna_message_info()**

```
void vrna_message_info (
            FILE * fp,
            const char * format,
             ...  )
```

#include <ViennaRNA/utils.h>

Print an info message.

This function is a wrapper to *fprintf*(...).

**See also**

vrna_message_vinfo(), vrna_message_error(), vrna_message_warning()

**Parameters**

| fp | The file pointer where the message is printed to |
|--------|---------------------------------------------------|
| format | The warning message to be printed |
| ... | Optional arguments for the formatted message string |

**15.45.3.12 vrna_message_vinfo()**

```
void vrna_message_vinfo (
            FILE * fp,
            const char * format,
            va_list args )
```

#include <ViennaRNA/utils.h>

Print an info message.

This function is a wrapper to *fprintf*(...).

**See also**

vrna_message_vinfo(), vrna_message_error(), vrna_message_warning()

**Parameters**

| fp | The file pointer where the message is printed to |
|--------|---------------------------------------------------|
| format | The info message to be printed |
| args | The argument list for the formatted message string |

**15.45.3.13 vrna_urn()**

```
double vrna_urn (
            void  )
```

#include <[ViennaRNA/utils.h](#)>

get a random number from [0..1]

**See also**

>   [vrna_int_urn(), vrna_init_rand()](#)

**Note**

>   Usually implemented by calling *erand48()*.

**Returns**

>   A random number in range [0..1]

**15.45.3.14 vrna_int_urn()**

```
int vrna_int_urn (
            int from,
            int to )
```

#include <[ViennaRNA/utils.h](#)>

Generates a pseudo random integer in a specified range.

**See also**

>   [vrna_urn(), vrna_init_rand()](#)

**Parameters**

| | |
|---|---|
| *from* | The first number in range |
| *to* | The last number in range |

**Returns**

>   A pseudo random number in range [from, to]

**15.45.3.15  vrna_time_stamp()**

```
char* vrna_time_stamp (
            void  )
```

`#include <`ViennaRNA/utils.h`>`

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

**15.45.3.16  get_input_line()**

```
unsigned int get_input_line (
            char ** string,
            unsigned int options )
```

`#include <`ViennaRNA/utils.h`>`

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 one to switch between different reading modes. Currently available options are:
#VRNA_INPUT_NOPRINT_COMMENTS, VRNA_INPUT_NOSKIP_COMMENTS, #VRNA_INPUT_NOELIM_W↩
S_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

**15.45.3.17    vrna_message_input_seq_simple()**

```
void vrna_message_input_seq_simple (
            void  )
```

#include <ViennaRNA/utils.h>

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

**15.45.3.18    vrna_message_input_seq()**

```
void vrna_message_input_seq (
            const char * s )
```

#include <ViennaRNA/utils.h>

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 |

**15.45.3.19    vrna_idx_row_wise()**

```
int* vrna_idx_row_wise (
            unsigned int length )
```

#include <ViennaRNA/utils.h>

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 ViennaRNA Package

Consult the implemented code to find out about the mapping formula ;)

**See also**

vrna_idx_col_wise()

**Parameters**

| length | The length of the RNA sequence |
|--------|--------------------------------|

**Returns**

The mapper array

**15.45.3.20   vrna_idx_col_wise()**

```
int* vrna_idx_col_wise (
              unsigned int length )
```

`#include <`ViennaRNA/utils.h`>`

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**

vrna_idx_row_wise()

**Parameters**

| length | The length of the RNA sequence |
|--------|--------------------------------|

**Returns**

The mapper array

**15.45.4   Variable Documentation**

**15.45.4.1   xsubi**

```
unsigned short xsubi[3]
```

```
#include <ViennaRNA/utils.h>
```

Current 48 bit random number.

This variable is used by vrna_urn(). These should be set to some random number seeds before the first call to vrna_urn().

**See also**

vrna_urn()

```
#include <ViennaRNA/utils.h>
```

## 15.46 MFE Structures of single Nucleic Acid Sequences

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

### 15.46.1 Detailed Description

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

Collaboration diagram for MFE Structures of single Nucleic Acid Sequences:

**Functions**

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

  *Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.*

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

  *Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.*

- float fold_par (const char ∗sequence, char ∗structure, vrna_param_t ∗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.*

- void update_fold_params_par (vrna_param_t ∗parameters)

  *Recalculate energy parameters.*

- void export_fold_arrays (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p)
- void export_fold_arrays_par (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p, vrna_param_t ∗∗P_p)
- void export_circfold_arrays (int ∗Fc_p, int ∗FcH_p, int ∗FcI_p, int ∗FcM_p, int ∗∗fM2_p, int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p)
- void export_circfold_arrays_par (int ∗Fc_p, int ∗FcH_p, int ∗FcI_p, int ∗FcM_p, int ∗∗fM2_p, int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p, vrna_param_t ∗∗P_p)
- 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)

### 15.46.2 Function Documentation

**15.46.2.1 vrna_fold()**

```
float vrna_fold (
            const char * sequence,
            char * structure )
```

`#include <`ViennaRNA/fold.h`>`

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.

This simplified interface to vrna_mfe() computes the MFE and, if required, a secondary structure for an RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_circfold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

**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

**15.46.2.2 vrna_circfold()**

```
float vrna_circfold (
            const char * sequence,
            char * structure )
```

`#include <`ViennaRNA/fold.h`>`

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.

This simplified interface to vrna_mfe() computes the MFE and, if required, a secondary structure for a circular RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [9] for further details.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_fold(), vrna_mfe(), vrna_fold_compound(), vrna_fold_compound_t

**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

**15.46.2.3 fold_par()**

```
float fold_par (
            const char * sequence,
            char * structure,
            vrna_param_t * parameters,
            int is_constrained,
            int is_circular )
```

#include <ViennaRNA/fold.h>

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 $\Sigma$ 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 $\mathrm{strlen(sequence)} + 1$

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

**Deprecated** use vrna_mfe() instead!

**See also**

> vrna_mfe(), fold(), circfold(), vrna_md_t, 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 pre-scaled energy contributions and the model details. (NULL may be passed, see OpenMP notes above) |
| *is_constrained* | Switch to indicate that a structure constraint is passed via the structure argument (0==off) |
| *is_circular* | Switch to (de-)activate post-processing steps in case RNA sequence is circular (0==off) |

**Returns**

> the minimum free energy (MFE) in kcal/mol

**15.46.2.4 fold()**

```
float fold (
            const char * sequence,
            char * structure )
```

```
#include <ViennaRNA/fold.h>
```

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

**Deprecated** use vrna_fold(), or vrna_mfe() instead!

**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

**15.46.2.5 circfold()**

```
float circfold (
            const char * sequence,
            char * structure )
```

#include <ViennaRNA/fold.h>

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

**Deprecated** Use vrna_circfold(), or vrna_mfe() instead!

**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

**15.46.2.6 free_arrays()**

```
void free_arrays (
            void )
```

#include <ViennaRNA/fold.h>

Free arrays for mfe folding.

**Deprecated** See vrna_fold(), vrna_circfold(), or vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**15.46.2.7 update_fold_params()**

```
void update_fold_params (
            void  )
```

#include <ViennaRNA/fold.h>

Recalculate energy parameters.

**Deprecated** For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

**15.46.2.8 update_fold_params_par()**

```
void update_fold_params_par (
            vrna_param_t * parameters )
```

#include <ViennaRNA/fold.h>

Recalculate energy parameters.

**Deprecated** For non-default model settings use the new API with vrna_params_subst() and vrna_mfe() instead!

**15.46.2.9 export_fold_arrays()**

```
void export_fold_arrays (
            int ** f5_p,
            int ** c_p,
            int ** fML_p,
            int ** fM1_p,
            int ** indx_p,
            char ** ptype_p )
```

#include <ViennaRNA/fold.h>

**Deprecated** See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**15.46.2.10 export_fold_arrays_par()**

```
void export_fold_arrays_par (
            int ** f5_p,
            int ** c_p,
            int ** fML_p,
            int ** fM1_p,
            int ** indx_p,
            char ** ptype_p,
            vrna_param_t ** P_p )
```

#include <ViennaRNA/fold.h>

**Deprecated** See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**15.46.2.11 export_circfold_arrays()**

```
void export_circfold_arrays (
            int * Fc_p,
            int * FcH_p,
            int * FcI_p,
            int * FcM_p,
            int ** fM2_p,
            int ** f5_p,
            int ** c_p,
            int ** fML_p,
            int ** fM1_p,
            int ** indx_p,
            char ** ptype_p )
```

#include <ViennaRNA/fold.h>

**Deprecated** See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**15.46.2.12 export_circfold_arrays_par()**

```
void export_circfold_arrays_par (
            int * Fc_p,
            int * FcH_p,
            int * FcI_p,
            int * FcM_p,
            int ** fM2_p,
            int ** f5_p,
            int ** c_p,
            int ** fML_p,
            int ** fM1_p,
            int ** indx_p,
            char ** ptype_p,
            vrna_param_t ** P_p )
```

#include <ViennaRNA/fold.h>

**Deprecated** See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

**15.46.2.13   LoopEnergy()**

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

#include <ViennaRNA/fold.h>

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

**15.46.2.14   HairpinE()**

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

#include <ViennaRNA/fold.h>

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

**15.46.2.15   initialize_fold()**

```
void initialize_fold (
            int length )
```

#include <ViennaRNA/fold.h>

Allocate arrays for folding

**Deprecated** See vrna_mfe() and vrna_fold_compound_t for the usage of the new API!

```
            int type_2,
```

## 15.47   MFE Structures of two hybridized Sequences

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.

### 15.47.1   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.

Collaboration diagram for MFE Structures of two hybridized Sequences:

**Functions**

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

  *Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.*
- 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, vrna_param_t ∗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 update_cofold_params_par (vrna_param_t ∗parameters)

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

  *get_monomer_free_energies*
- void initialize_cofold (int length)
- float vrna_mfe_dimer (vrna_fold_compound_t ∗vc, char ∗structure)

  *Compute the minimum free energy of two interacting RNA molecules.*

### 15.47.2   Function Documentation

**15.47.2.1 vrna_cofold()**

```
float vrna_cofold (
            const char * sequence,
            char * structure )
```

```
#include <ViennaRNA/cofold.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.

This simplified interface to vrna_mfe() computes the MFE and, if required, a secondary structure for two RNA sequences upon dimerization using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe(), and the data structure vrna_fold_compound_t instead.

**See also**

vrna_mfe_dimer(), vrna_fold_compound(), vrna_fold_compound_t, vrna_cut_point_insert()

**Parameters**

| sequence | two RNA sequences separated by the '&' character |
| --- | --- |
| 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

**15.47.2.2 cofold()**

```
float cofold (
            const char * sequence,
            char * structure )
```

```
#include <ViennaRNA/cofold.h>
```

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().

**Deprecated** use vrna_mfe_dimer() instead

**Parameters**

| | |
|---|---|
| *sequence* | The two sequences concatenated |
| *structure* | Will hold the barcket dot structure of the dimer molecule |

**Returns**

minimum free energy of the structure

**15.47.2.3   cofold_par()**

```
float cofold_par (
            const char * string,
            char * structure,
            vrna_param_t * parameters,
            int is_constrained )
```

#include <ViennaRNA/cofold.h>

Compute the minimum free energy of two interacting RNA molecules.

**Deprecated** use vrna_mfe_dimer() instead

**15.47.2.4   free_co_arrays()**

```
void free_co_arrays (
            void  )
```

#include <ViennaRNA/cofold.h>

Free memory occupied by cofold()

**Deprecated** This function will only free memory allocated by a prior call of cofold() or cofold_par(). See vrna_mfe_dimer() for how to use the new API

**Note**

folding matrices now reside in the fold compound, and should be free'd there

**See also**

vrna_fc_destroy(), vrna_mfe_dimer()

**15.47.2.5   update_cofold_params()**

```
void update_cofold_params (
            void  )
```

#include <ViennaRNA/cofold.h>

Recalculate parameters.

**Deprecated**  See vrna_params_subst() for an alternative using the new API

**15.47.2.6   update_cofold_params_par()**

```
void update_cofold_params_par (
            vrna_param_t * parameters )
```

#include <ViennaRNA/cofold.h>

Recalculate parameters.

**Deprecated**  See vrna_params_subst() for an alternative using the new API

**15.47.2.7   export_cofold_arrays_gq()**

```
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 )
```

#include <ViennaRNA/cofold.h>

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 back-tracking

**Deprecated**  folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to cofold() or cofold_par()

**See also**

vrna_mfe_dimer() for the new API

**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) |

**15.47.2.8 export_cofold_arrays()**

```
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 )
```

#include <ViennaRNA/cofold.h>

Export the arrays of partition function cofold.

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

**Deprecated** folding matrices now reside within the vrna_fold_compound_t. Thus, this function will only work in conjunction with a prior call to the deprecated functions cofold() or cofold_par()

**See also**

vrna_mfe_dimer() for the new API

**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) |

**15.47.2.9  get_monomere_mfes()**

```
void get_monomere_mfes (
            float * e1,
            float * e2 )
```

#include <ViennaRNA/cofold.h>

get_monomer_free_energies

Export monomer free energies out of cofold arrays

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

**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 |

**15.47.2.10  initialize_cofold()**

```
void initialize_cofold (
            int length )
```

#include <ViennaRNA/cofold.h>

allocate arrays for folding

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

**15.47.2.11  vrna_mfe_dimer()**

```
float vrna_mfe_dimer (
            vrna_fold_compound_t * vc,
            char * structure )
```

#include <ViennaRNA/mfe.h>

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the vrna_mfe() function.

**Parameters**

| | |
|---|---|
| *vc* | fold compound |
| *structure* | Will hold the barcket dot structure of the dimer molecule |

**Returns**

minimum free energy of the structure

**SWIG Wrapper Notes**  This function is attached as method **mfe_dimer()** to objects of type *fold_compound*

## 15.48 Partition Function for two hybridized Sequences

Partition Function Cofolding.

### 15.48.1 Detailed Description

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 [2] for further details.

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) Collaboration diagram for Partition Function for two hybridized Sequences:

#### Files

- file concentrations.h

  *Concentration computations for RNA-RNA interactions.*

- file part_func_co.h

  *Partition function for two RNA sequences.*

- file part_func_up.h

  *Implementations for accessibility and RNA-RNA interaction as a stepwise process.*

#### Data Structures

- struct vrna_dimer_pf_s

  *Data structure returned by vrna_pf_dimer() More...*

#### Typedefs

- typedef struct vrna_dimer_pf_s vrna_dimer_pf_t

  *Typename for the data structure that stores the dimer partition functions, vrna_dimer_pf_s, as returned by vrna_pf_dimer()*

- typedef struct vrna_dimer_pf_s cofoldF

  *Backward compatibility typedef for vrna_dimer_pf_s.*

## Functions

- void vrna_pf_dimer_probs (double FAB, double FA, double FB, vrna_ep_t ∗prAB, const vrna_ep_t ∗prA, const vrna_ep_t ∗prB, int Alength, const vrna_exp_param_t ∗exp_params)

    *Compute Boltzmann probabilities of dimerization without homodimers.*

- vrna_dimer_pf_t vrna_pf_dimer (vrna_fold_compound_t ∗vc, char ∗structure)

    *Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

- vrna_dimer_pf_t vrna_pf_co_fold (const char ∗seq, char ∗structure, vrna_ep_t ∗∗pl)

    *Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

## Variables

- int mirnatog

    *Toggles no intrabp in 2nd mol.*

- double F_monomer [2]

    *Free energies of the two monomers.*

- typedef struct vrna_dimer_conc_s vrna_dimer_conc_t

    *Typename for the data structure that stores the dimer concentrations, vrna_dimer_conc_s, as required by vrna_pf↩ _dimer_concentration()*

- typedef struct vrna_dimer_conc_s ConcEnt

    *Backward compatibility typedef for vrna_dimer_conc_s.*

- vrna_dimer_conc_t ∗ vrna_pf_dimer_concentrations (double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double ∗startconc, const vrna_exp_param_t ∗exp_params)

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

## 15.48.2 Data Structure Documentation

### 15.48.2.1 struct vrna_dimer_pf_s

Data structure returned by vrna_pf_dimer()

**Data Fields**

- double F0AB

    *Null model without DuplexInit.*

- double FAB

    *all states with DuplexInit correction*

- double FcAB

    *true hybrid states only*

- double FA

    *monomer A*

- double FB

    *monomer B*

## 15.48.3 Function Documentation

### 15.48.3.1 vrna_pf_dimer_probs()

```
void vrna_pf_dimer_probs (
            double FAB,
            double FA,
            double FB,
            vrna_ep_t * prAB,
            const vrna_ep_t * prA,
            const vrna_ep_t * prB,
            int Alength,
            const vrna_exp_param_t * exp_params )
```

#include <ViennaRNA/equilibrium_probs.h>

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 vrna_plist_from_probs(), the dimer probabilities 'prAB' are modified in place.

**Parameters**

| | |
|---|---|
| FAB | free energy of dimer AB |
| FA | free energy of monomer A |
| FB | free energy of monomer B |
| prAB | pair probabilities for dimer |
| prA | pair probabilities monomer |
| prB | pair probabilities monomer |
| Alength | Length of molecule A |
| exp_params | The precomputed Boltzmann factors |

### 15.48.3.2 vrna_pf_dimer()

```
vrna_dimer_pf_t vrna_pf_dimer (
            vrna_fold_compound_t * vc,
            char * structure )
```

#include <ViennaRNA/part_func.h>

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This is the cofold partition function folding.

**See also**

> vrna_fold_compound() for how to retrieve the necessary data structure

**Parameters**

| vc | the fold compound data structure |
|---|---|
| *structure* | Will hold the structure or constraints |

**Returns**

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

**SWIG Wrapper Notes** This function is attached as method **pf_dimer()** to objects of type *fold_compound*

**15.48.3.3 vrna_pf_co_fold()**

vrna_dimer_pf_t vrna_pf_co_fold (
          const char * *seq,*
          char * *structure,*
          vrna_ep_t ** *pl* )

#include <ViennaRNA/part_func_co.h>

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This simplified interface to vrna_pf_dimer() computes the partition function and, if required, base pair probabilities for an RNA-RNA interaction using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_pf_dimer(), and the data structure vrna_fold_compound_t instead.

**See also**

vrna_pf_dimer()

**Parameters**

| seq | Two concatenated RNA sequences with a delimiting '&' in between |
|---|---|
| *structure* | A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL) |
| *pl* | A pointer to a list of vrna_ep_t to store pairing probabilities (Maybe NULL) |

**Returns**

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

**15.48.3.4 vrna_pf_dimer_concentrations()**

vrna_dimer_conc_t* vrna_pf_dimer_concentrations (
          double *FcAB,*
          double *FcAA,*
          double *FcBB,*
          double *FEA,*
          double *FEB,*
          const double * *startconc,*
          const vrna_exp_param_t * *exp_params* )

#include <ViennaRNA/concentrations.h>

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 vrna_dimer_pf_t struct.

**Parameters**

| | |
|---|---|
| *FcAB* | Free energy of AB dimer (FcAB entry) |
| *FcAA* | Free energy of AA dimer (FcAB entry) |
| *FcBB* | 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] |
| *exp_params* | The precomputed Boltzmann factors |

**Returns**

    vrna_dimer_conc_t array containing the equilibrium energies and start concentrations

## 15.49 Partition Function for two hybridized Sequences as a stepwise Process

RNA-RNA interaction as a stepwise process.

### 15.49.1 Detailed Description

RNA-RNA interaction as a 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. Collaboration diagram for Partition Function for two hybridized Sequences as a 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().*

### 15.49.2 Function Documentation

#### 15.49.2.1 pf_unstru()

```
pu_contrib* pf_unstru (
            char * sequence,
            int max_w )
```

#include <ViennaRNA/part_func_up.h>

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**

**15.49.2.2  pf_interact()**

```
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 )
```

#include <ViennaRNA/part_func_up.h>

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**

## 15.50 Local MFE Structure Prediction and Z-scores

### 15.50.1 Detailed Description

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

### Functions

- float vrna_Lfold (const char ∗string, int window_size, FILE ∗file)

  *Local MFE prediction using a sliding window approach (simplified interface)*

- float vrna_Lfoldz (const char ∗string, int window_size, double min_z, FILE ∗file)

  *Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)*

- 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 vrna_mfe_window (vrna_fold_compound_t ∗vc, FILE ∗file)

  *Local MFE prediction using a sliding window approach.*

- float vrna_mfe_window_zscore (vrna_fold_compound_t ∗vc, double min_z, FILE ∗file)

  *Local MFE prediction using a sliding window approach (with z-score cut-off)*

### 15.50.2 Function Documentation

#### 15.50.2.1 vrna_Lfold()

```
float vrna_Lfold (
            const char * string,
            int window_size,
            FILE * file )
```

`#include <ViennaRNA/Lfold.h>`

Local MFE prediction using a sliding window approach (simplified interface)

This simplified interface to vrna_mfe_window() computes the MFE and locally optimal secondary structure using default options. Structures are predicted using a sliding window approach, where base pairs may not span outside the window. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe_window(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_mfe_window(), vrna_Lfoldz(), vrna_mfe_window_zscore(), vrna_fold_compound(), vrna_fold_compound_t

**Parameters**

| *string* | The nucleic acid sequence |
|---|---|
| *window_size* | The window size for locally optimal structures |
| *file* | The output file handle where predictions are written to (if NULL, output is written to stdout) |

**15.50.2.2   vrna_Lfoldz()**

```
float vrna_Lfoldz (
            const char * string,
            int window_size,
            double min_z,
            FILE * file )
```

`#include <`ViennaRNA/Lfold.h`>`

Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)

This simplified interface to vrna_mfe_window_zscore() computes the MFE and locally optimal secondary structure using default options. Structures are predicted using a sliding window approach, where base pairs may not span outside the window. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing. This function is the z-score version of vrna_Lfold(), i.e. only predictions above a certain z-score cut-off value are printed.

**Note**

> In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use vrna_mfe_window(), and the data structure vrna_fold_compound_t instead.

**See also**

> vrna_mfe_window_zscore(), vrna_Lfold(), vrna_mfe_window(), vrna_fold_compound(), vrna_fold_compound_t

**Parameters**

| *string* | The nucleic acid sequence |
|---|---|
| *window_size* | The window size for locally optimal structures |
| *min_z* | The minimal z-score for a predicted structure to appear in the output |
| *file* | The output file handle where predictions are written to (if NULL, output is written to stdout) |

**15.50.2.3   Lfold()**

```
float Lfold (
            const char * string,
```

```
            char * structure,
            int maxdist )
```

`#include <ViennaRNA/Lfold.h>`

The local analog to fold().

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

**Deprecated** Use vrna_mfe_window() instead!

### 15.50.2.4 Lfoldz()

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

`#include <ViennaRNA/Lfold.h>`

**Deprecated** Use vrna_mfe_window_zscore() instead!

### 15.50.2.5 vrna_mfe_window()

```
float vrna_mfe_window (
            vrna_fold_compound_t * vc,
            FILE * file )
```

`#include <ViennaRNA/mfe.h>`

Local MFE prediction using a sliding window approach.

Computes minimum free energy structures using a sliding window approach, where base pairs may not span outside the window. In contrast to vrna_mfe(), where a maximum base pair span may be set using the vrna_md_t.max_bp_span attribute and one globally optimal structure is predicted, this function uses a sliding window to retrieve all locally optimal structures within each window. The size of the sliding window is set in the vrna_md_t.window_size attribute, prior to the retrieval of the vrna_fold_compound_t using vrna_fold_compound() with option VRNA_OPTION_WINDOW

The predicted structures are written on-the-fly, either to stdout, if a NULL pointer is passed as file parameter, or to the corresponding filehandle.

**See also**

> vrna_fold_compound(), vrna_mfe_window_zscore(), vrna_mfe(), vrna_Lfold(), vrna_Lfoldz(), VRNA_OPTION_WINDOW, vrna_md_t.max_bp_span, vrna_md_t.window_size

**Parameters**

| vc | The vrna_fold_compound_t with preallocated memory for the DP matrices |
|---|---|
| file | The output file handle where predictions are written to (maybe NULL) |

**SWIG Wrapper Notes** This function is attached as method **mfe_window()** to objects of type *fold_compound*

**15.50.2.6 vrna_mfe_window_zscore()**

```
float vrna_mfe_window_zscore (
            vrna_fold_compound_t * vc,
            double min_z,
            FILE * file )
```

```
#include <ViennaRNA/mfe.h>
```

Local MFE prediction using a sliding window approach (with z-score cut-off)

Computes minimum free energy structures using a sliding window approach, where base pairs may not span outside the window. This function is the z-score version of vrna_mfe_window(), i.e. only predictions above a certain z-score cut-off value are printed. As for vrna_mfe_window(), the size of the sliding window is set in the vrna_md_t.window_size attribute, prior to the retrieval of the vrna_fold_compound_t using vrna_fold_compound() with option VRNA_OPTION_WINDOW.

The predicted structures are written on-the-fly, either to stdout, if a NULL pointer is passed as file parameter, or to the corresponding filehandle.

**See also**

vrna_fold_compound(), vrna_mfe_window_zscore(), vrna_mfe(), vrna_Lfold(), vrna_Lfoldz(), VRNA_OPTION_WINDOW, vrna_md_t.max_bp_span, vrna_md_t.window_size

**Parameters**

| vc | The vrna_fold_compound_t with preallocated memory for the DP matrices |
|---|---|
| min↩ _z | The minimal z-score for a predicted structure to appear in the output |
| file | The output file handle where predictions are written to (maybe NULL) |

## 15.51 Partition Function and Equilibrium Probabilities for Locally Stable Secondary Structures

### 15.51.1 Detailed Description

Collaboration diagram for Partition Function and Equilibrium Probabilities for Locally Stable Secondary Structures:

### Macros

- #define VRNA_PROBS_WINDOW_BPP 4096U

  *Option flag to activate callback triggers for pairing probabilities.*
- #define VRNA_PROBS_WINDOW_UP 8192U

  *Option flag to activate callback triggers for unpaired probabilities.*
- #define VRNA_PROBS_WINDOW_STACKP 16384U

  *Option flag to activate callback triggers for stacking probabilities.*
- #define VRNA_PROBS_WINDOW_UP_SPLIT 32768U

  *Option flag to activate callback triggers for unpaired probabilities (split up into different loop types)*

### Functions

- vrna_ep_t ∗ vrna_pfl_fold (const char ∗sequence, int window_size, int max_bp_span, float cutoff)

  *Compute base pair probabilities using a sliding-window approach.*
- void vrna_pfl_fold_cb (const char ∗sequence, int window_size, int max_bp_span, vrna_probs_window_callback ∗cb, void ∗data)

  *Compute base pair probabilities using a sliding-window approach (callback version)*
- double ∗∗ vrna_pfl_fold_up (const char ∗sequence, int ulength, int window_size, int max_bp_span)

  *Compute probability of contiguous unpaired segments.*
- void vrna_pfl_fold_up_cb (const char ∗sequence, int ulength, int window_size, int max_bp_span, vrna_probs_window_callback ∗cb, void ∗data)

  *Compute probability of contiguous unpaired segments.*
- void vrna_probs_window (vrna_fold_compound_t ∗fc, int ulength, unsigned int options, vrna_probs_window_callback ∗cb, void ∗data)

  *Compute various equilibrium probabilities under a sliding window approach.*
- void update_pf_paramsLP (int length)
- vrna_ep_t ∗ pfl_fold (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, vrna_ep_t ∗∗dpp2, FILE ∗pUfp, FILE ∗spup)

  *Compute partition functions for locally stable secondary structures.*
- vrna_ep_t ∗ pfl_fold_par (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, vrna_ep_t ∗∗dpp2, FILE ∗pUfp, FILE ∗spup, vrna_exp_param_t ∗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.*

## 15.51.2 Macro Definition Documentation

### 15.51.2.1 VRNA_PROBS_WINDOW_BPP

```
#define VRNA_PROBS_WINDOW_BPP 4096U
```

```
#include <ViennaRNA/LPfold.h>
```

Option flag to activate callback triggers for pairing probabilities.

**See also**

vrna_probs_window()

### 15.51.2.2 VRNA_PROBS_WINDOW_UP

```
#define VRNA_PROBS_WINDOW_UP 8192U
```

```
#include <ViennaRNA/LPfold.h>
```

Option flag to activate callback triggers for unpaired probabilities.

**See also**

vrna_probs_window()

### 15.51.2.3 VRNA_PROBS_WINDOW_STACKP

```
#define VRNA_PROBS_WINDOW_STACKP 16384U
```

```
#include <ViennaRNA/LPfold.h>
```

Option flag to activate callback triggers for stacking probabilities.

**See also**

vrna_probs_window()

**15.51.2.4 VRNA_PROBS_WINDOW_UP_SPLIT**

```
#define VRNA_PROBS_WINDOW_UP_SPLIT 32768U
```

`#include <`ViennaRNA/LPfold.h`>`

Option flag to activate callback triggers for unpaired probabilities (split up into different loop types)

**See also**

vrna_probs_window(), #VRNA_EXT_LOOP, #VRNA_HP_LOOP, #VRNA_INT_LOOP, #VRNA_MB_LOOP, #VRNA_ANY_LOOP

**15.51.3 Function Documentation**

**15.51.3.1 vrna_pfl_fold()**

```
vrna_ep_t* vrna_pfl_fold (
            const char * sequence,
            int window_size,
            int max_bp_span,
            float cutoff )
```

`#include <`ViennaRNA/LPfold.h`>`

Compute base pair probabilities using a sliding-window approach.

This is a simplified wrapper to vrna_probs_window() that given a nucleid acid sequence, a window size, a maximum base pair span, and a cutoff value computes the pair probabilities for any base pair in any window. The pair probabilities are returned as a list and the user has to take care to free() the memory occupied by the list.

**Note**

This function uses default model settings! For custom model settings, we refer to the function vrna_probs_window().

**See also**

vrna_probs_window(), vrna_pfl_fold_cb(), vrna_pfl_fold_up()

**Parameters**

| | |
|---|---|
| *sequence* | The nucleic acid input sequence |
| *window_size* | The size of the sliding window |
| *max_bp_span* | The maximum distance along the backbone between two nucleotides that form a base pairs |
| *cutoff* | A cutoff value that omits all pairs with lower probability |

**Returns**

A list of base pair probabilities, terminated by an entry with #vrna_ep_t.i and #vrna_ep_t.j set to 0

**15.51.3.2   vrna_pfl_fold_cb()**

```
void vrna_pfl_fold_cb (
            const char * sequence,
            int window_size,
            int max_bp_span,
            vrna_probs_window_callback * cb,
            void * data )
```

`#include <`ViennaRNA/LPfold.h`>`

Compute base pair probabilities using a sliding-window approach (callback version)

This is a simplified wrapper to vrna_probs_window() that given a nucleid acid sequence, a window size, a maximum base pair span, and a cutoff value computes the pair probabilities for any base pair in any window. It is similar to vrna_pfl_fold() but uses a callback mechanism to return the pair probabilities.

Read the details for vrna_probs_window() for details on the callback implementation!

**Note**

This function uses default model settings!   For custom model settings, we refer to the function vrna_probs_window().

**See also**

vrna_probs_window(), vrna_pfl_fold(), vrna_pfl_fold_up_cb()

**Parameters**

| sequence | The nucleic acid input sequence |
|---|---|
| window_size | The size of the sliding window |
| max_bp_span | The maximum distance along the backbone between two nucleotides that form a base pairs |
| cb | The callback function which collects the pair probability data for further processing |
| data | Some arbitrary data structure that is passed to the callback `cb` |

**15.51.3.3   vrna_pfl_fold_up()**

```
double** vrna_pfl_fold_up (
            const char * sequence,
            int ulength,
```

```
        int window_size,
        int max_bp_span )
```

#include <ViennaRNA/LPfold.h>

Compute probability of contiguous unpaired segments.

This is a simplified wrapper to vrna_probs_window() that given a nucleic acid sequence, a maximum length of unpaired segments (ulength), a window size, and a maximum base pair span computes the equilibrium probability of any segment not exceeding ulength. The probabilities to be unpaired are returned as a 1-based, 2-dimensional matrix with dimensions $N \times M$, where $N$ is the length of the sequence and $M$ is the maximum segment length. As an example, the probability of a segment of size 5 starting at position 100 is stored in the matrix entry $X[100][5]$.

It is the users responsibility to free the memory occupied by this matrix.

**Note**

> This function uses default model settings! For custom model settings, we refer to the function vrna_probs_window().

**Parameters**

| | |
|---|---|
| *sequence* | The nucleic acid input sequence |
| *ulength* | The maximal length of an unpaired segment |
| *window_size* | The size of the sliding window |
| *max_bp_span* | The maximum distance along the backbone between two nucleotides that form a base pairs |

**Returns**

> The probabilities to be unpaired for any segment not exceeding ulength

**15.51.3.4 vrna_pfl_fold_up_cb()**

```
void vrna_pfl_fold_up_cb (
        const char * sequence,
        int ulength,
        int window_size,
        int max_bp_span,
        vrna_probs_window_callback * cb,
        void * data )
```

#include <ViennaRNA/LPfold.h>

Compute probability of contiguous unpaired segments.

This is a simplified wrapper to vrna_probs_window() that given a nucleic acid sequence, a maximum length of unpaired segments (ulength), a window size, and a maximum base pair span computes the equilibrium probability of any segment not exceeding ulength. It is similar to vrna_pfl_fold_up() but uses a callback mechanism to return the unpaired probabilities.

Read the details for vrna_probs_window() for details on the callback implementation!

**Note**

> This function uses default model settings! For custom model settings, we refer to the function vrna_probs_window().

**Parameters**

| sequence | The nucleic acid input sequence |
|---|---|
| ulength | The maximal length of an unpaired segment |
| window_size | The size of the sliding window |
| max_bp_span | The maximum distance along the backbone between two nucleotides that form a base pairs |
| cb | The callback function which collects the pair probability data for further processing |
| data | Some arbitrary data structure that is passed to the callback `cb` |

**Returns**

The probabilities to be unpaired for any segment not exceeding `ulength`

**15.51.3.5   vrna_probs_window()**

```
void vrna_probs_window (
              vrna_fold_compound_t * fc,
              int ulength,
              unsigned int options,
              vrna_probs_window_callback * cb,
              void * data )
```

`#include <ViennaRNA/LPfold.h>`

Compute various equilibrium probabilities under a sliding window approach.

**See also**

vrna_pfl_fold_cb(), vrna_pfl_fold_up_cb(), VRNA_PROBS_WINDOW_BPP, VRNA_PROBS_WINDOW_UP,
VRNA_PROBS_WINDOW_UP_SPLIT

**Parameters**

| fc | The fold compound with sequence data, model settings and precomputed energy parameters |
|---|---|
| ulength | The maximal length of an unpaired segment (only for unpaired probability computations) |
| cb | The callback function which collects the pair probability data for further processing |
| data | Some arbitrary data structure that is passed to the callback `cb` |
| options | Option flags to control the behavior of this function |

**15.51.3.6   update_pf_paramsLP()**

```
void update_pf_paramsLP (
              int length )
```

`#include <ViennaRNA/LPfold.h>`

**Parameters**

| length | |
| --- | --- |

**15.51.3.7 pfl_fold()**

```
vrna_ep_t* pfl_fold (
            char * sequence,
            int winSize,
            int pairSize,
            float cutoffb,
            double ** pU,
            vrna_ep_t ** dpp2,
            FILE * pUfp,
            FILE * spup )
```

`#include <ViennaRNA/LPfold.h>`

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

### 15.51.3.8 putoutpU_prob()

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

#include <ViennaRNA/LPfold.h>

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 |

### 15.51.3.9 putoutpU_prob_bin()

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

#include <ViennaRNA/LPfold.h>

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 |

## 15.52 Local MFE Consensus Structure Prediction for Sequence Alignments

### 15.52.1 Detailed Description

Collaboration diagram for Local MFE Consensus Structure Prediction for Sequence Alignments:

**Functions**

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

### 15.52.2 Function Documentation

#### 15.52.2.1 aliLfold()

```
float aliLfold (
            const char ** AS,
            char * structure,
            int maxdist )
```

#include <ViennaRNA/Lfold.h>

**Parameters**

| strings | |
|---|---|
| structure | |
| maxdist | |

**Returns**

## 15.53 Reading/Writing Energy Parameter Sets from/to File

Read and Write energy parameter sets from and to text files.

### 15.53.1 Detailed Description

Read and Write energy parameter sets from and to text files.

A default set of parameters, identical to the one described in [14] and [19], is compiled into the library. Collaboration diagram for Reading/Writing Energy Parameter Sets from/to File:

**Modules**

- Converting Energy Parameter Files

    *Convert energy parameter files into the latest format.*

**Functions**

- const char ∗ last_parameter_file (void)

    *Get the file name of the parameter file that was most recently loaded.*
- 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.*

### 15.53.2 Function Documentation

#### 15.53.2.1 last_parameter_file()

```
const char* last_parameter_file (
            void  )
```

#include <ViennaRNA/read_epars.h>

Get the file name of the parameter file that was most recently loaded.

**Returns**

The file name of the last parameter file, or NULL if parameters are still at defaults

#### 15.53.2.2 read_parameter_file()

```
void read_parameter_file (
            const char fname[ ] )
```

#include <ViennaRNA/read_epars.h>

Read energy parameters from a file.

**Parameters**

| | |
|---|---|
| *fname* | The path to the file containing the energy parameters |

**15.53.2.3   write_parameter_file()**

```
void write_parameter_file (
            const char fname[] )
```

#include <[ViennaRNA/read_epars.h](#)>

Write energy parameters to a file.

**Parameters**

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

## 15.54 Converting Energy Parameter Files

Convert energy parameter files into the latest format.

### 15.54.1 Detailed Description

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 Collaboration diagram for Converting Energy Parameter Files:

**Files**

- file 1.8.4_epars.h

  *Free energy parameters for parameter file conversion.*
- file 1.8.4_intloops.h

  *Free energy parameters for interior loop contributions needed by the parameter file conversion functions.*

**Macros**

- #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)

## 15.54.2 Macro Definition Documentation

### 15.54.2.1 VRNA_CONVERT_OUTPUT_ALL

#define VRNA_CONVERT_OUTPUT_ALL 1U

#include <ViennaRNA/convert_epars.h>

Flag to indicate printing of a complete parameter set

### 15.54.2.2 VRNA_CONVERT_OUTPUT_HP

#define VRNA_CONVERT_OUTPUT_HP 2U

#include <ViennaRNA/convert_epars.h>

Flag to indicate printing of hairpin contributions

### 15.54.2.3 VRNA_CONVERT_OUTPUT_STACK

#define VRNA_CONVERT_OUTPUT_STACK 4U

#include <ViennaRNA/convert_epars.h>

Flag to indicate printing of base pair stack contributions

### 15.54.2.4 VRNA_CONVERT_OUTPUT_MM_HP

#define VRNA_CONVERT_OUTPUT_MM_HP 8U

#include <ViennaRNA/convert_epars.h>

Flag to indicate printing of hairpin mismatch contribution

### 15.54.2.5 VRNA_CONVERT_OUTPUT_MM_INT

#define VRNA_CONVERT_OUTPUT_MM_INT 16U

#include <ViennaRNA/convert_epars.h>

Flag to indicate printing of interior loop mismatch contribution

**15.54.2.6   VRNA_CONVERT_OUTPUT_MM_INT_1N**

`#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U`

`#include <`ViennaRNA/convert_epars.h`>`

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

**15.54.2.7   VRNA_CONVERT_OUTPUT_MM_INT_23**

`#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U`

`#include <`ViennaRNA/convert_epars.h`>`

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

**15.54.2.8   VRNA_CONVERT_OUTPUT_MM_MULTI**

`#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U`

`#include <`ViennaRNA/convert_epars.h`>`

Flag to indicate printing of multi loop mismatch contribution

**15.54.2.9   VRNA_CONVERT_OUTPUT_MM_EXT**

`#define VRNA_CONVERT_OUTPUT_MM_EXT 256U`

`#include <`ViennaRNA/convert_epars.h`>`

Flag to indicate printing of exterior loop mismatch contribution

**15.54.2.10   VRNA_CONVERT_OUTPUT_DANGLE5**

`#define VRNA_CONVERT_OUTPUT_DANGLE5 512U`

`#include <`ViennaRNA/convert_epars.h`>`

Flag to indicate printing of 5' dangle conctribution

**15.54.2.11   VRNA_CONVERT_OUTPUT_DANGLE3**

`#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U`

`#include <`ViennaRNA/convert_epars.h`>`

Flag to indicate printing of 3' dangle contribution

**15.54.2.12  VRNA_CONVERT_OUTPUT_INT_11**

#define VRNA_CONVERT_OUTPUT_INT_11 2048U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of 1:1 interior loop contribution

**15.54.2.13  VRNA_CONVERT_OUTPUT_INT_21**

#define VRNA_CONVERT_OUTPUT_INT_21 4096U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of 2:1 interior loop contribution

**15.54.2.14  VRNA_CONVERT_OUTPUT_INT_22**

#define VRNA_CONVERT_OUTPUT_INT_22 8192U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of 2:2 interior loop contribution

**15.54.2.15  VRNA_CONVERT_OUTPUT_BULGE**

#define VRNA_CONVERT_OUTPUT_BULGE 16384U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of bulge loop contribution

**15.54.2.16  VRNA_CONVERT_OUTPUT_INT**

#define VRNA_CONVERT_OUTPUT_INT 32768U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of interior loop contribution

**15.54.2.17  VRNA_CONVERT_OUTPUT_ML**

#define VRNA_CONVERT_OUTPUT_ML 65536U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of multi loop contribution

### 15.54.2.18 VRNA_CONVERT_OUTPUT_MISC

#define VRNA_CONVERT_OUTPUT_MISC 131072U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

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

### 15.54.2.19 VRNA_CONVERT_OUTPUT_SPECIAL_HP

#define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

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

### 15.54.2.20 VRNA_CONVERT_OUTPUT_VANILLA

#define VRNA_CONVERT_OUTPUT_VANILLA 524288U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of given parameters only

**Note**

> This option overrides all other output options, except [VRNA_CONVERT_OUTPUT_DUMP](VRNA_CONVERT_OUTPUT_DUMP) !

### 15.54.2.21 VRNA_CONVERT_OUTPUT_NINIO

#define VRNA_CONVERT_OUTPUT_NINIO 1048576U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

Flag to indicate printing of interior loop asymmetry contribution

### 15.54.2.22 VRNA_CONVERT_OUTPUT_DUMP

#define VRNA_CONVERT_OUTPUT_DUMP 2097152U

#include <[ViennaRNA/convert_epars.h](ViennaRNA/convert_epars.h)>

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

### 15.54.3 Function Documentation

#### 15.54.3.1 convert_parameter_file()

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

#include <ViennaRNA/convert_epars.h>

Convert/dump a Vienna 1.8.4 formatted energy parameter file

The options argument allows one 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) |

## 15.55 The Fold Compound

This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.

### 15.55.1 Detailed Description

This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.

Throughout the entire RNAlib, the vrna_fold_compound_t, is used to group information and data that is required for structure prediction and energy evaluation. Here, you'll find interface functions to create, modify, and delete vrna_fold_compound_t data structures. Collaboration diagram for The Fold Compound:

**Files**

- file fold_compound.h

    *The Basic Fold Compound API.*

**Data Structures**

- struct vrna_fc_s

    *The most basic data structure required by many functions throughout the RNAlib. More...*

**Macros**

- #define VRNA_STATUS_MFE_PRE (unsigned char)1

    *Status message indicating that MFE computations are about to begin.*
- #define VRNA_STATUS_MFE_POST (unsigned char)2

    *Status message indicating that MFE computations are finished.*
- #define VRNA_STATUS_PF_PRE (unsigned char)3

    *Status message indicating that Partition function computations are about to begin.*
- #define VRNA_STATUS_PF_POST (unsigned char)4

    *Status message indicating that Partition function computations are finished.*
- #define VRNA_OPTION_DEFAULT 0U

    *Option flag to specify default settings/requirements.*
- #define VRNA_OPTION_MFE 1U

    *Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.*
- #define VRNA_OPTION_PF 2U

    *Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.*
- #define VRNA_OPTION_HYBRID 4U

    *Option flag to specify requirement of dimer DP matrices.*
- #define VRNA_OPTION_EVAL_ONLY 8U

    *Option flag to specify that neither MFE, nor PF DP matrices are required.*
- #define VRNA_OPTION_WINDOW 16U

    *Option flag to specify requirement of DP matrices for local folding approaches.*

**Typedefs**

- typedef struct vrna_fc_s vrna_fold_compound_t

    *Typename for the fold_compound data structure vrna_fc_s.*

- typedef void() vrna_callback_free_auxdata(void *data)

    *Callback to free memory allocated for auxiliary user-provided data.*

- typedef void() vrna_callback_recursion_status(unsigned char status, void *data)

    *Callback to perform specific user-defined actions before, or after recursive computations.*

**Enumerations**

- enum vrna_fc_type_e { VRNA_FC_TYPE_SINGLE, VRNA_FC_TYPE_COMPARATIVE }

    *An enumerator that is used to specify the type of a vrna_fold_compound_t.*

**Functions**

- vrna_fold_compound_t ∗ vrna_fold_compound (const char ∗sequence, vrna_md_t ∗md_p, unsigned int options)

    *Retrieve a vrna_fold_compound_t data structure for single sequences and hybridizing sequences.*

- vrna_fold_compound_t ∗ vrna_fold_compound_comparative (const char ∗∗sequences, vrna_md_t ∗md_p, unsigned int options)

    *Retrieve a vrna_fold_compound_t data structure for sequence alignments.*

- void vrna_fold_compound_free (vrna_fold_compound_t ∗vc)

    *Free memory occupied by a vrna_fold_compound_t.*

- void vrna_fold_compound_add_auxdata (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗f)

    *Add auxiliary data to the vrna_fold_compound_t.*

- void vrna_fold_compound_add_callback (vrna_fold_compound_t ∗vc, vrna_callback_recursion_status ∗f)

    *Add a recursion status callback to the vrna_fold_compound_t.*

### 15.55.2 Data Structure Documentation

#### 15.55.2.1 struct vrna_fc_s

The most basic data structure required by many functions throughout the RNAlib.

**Note**

Please read the documentation of this data structure carefully! Some attributes are only available for specific types this data structure can adopt.

**Warning**

Reading/Writing from/to attributes that are not within the scope of the current type usually result in undefined behavior!

**See also**

vrna_fold_compound_t.type, vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), VRNA_FC_TYPE_SINGLE, VRNA_FC_TYPE_COMPARATIVE

**SWIG Wrapper Notes** This data structure is wrapped as an object **fold_compound** with several related functions attached as methods.

A new **fold_compound** can be obtained by calling one of its constructors:

- *fold_compound(seq)* – Initialize with a single sequence, or two concatenated sequences separated by an ampersand character '&' (for cofolding)
- *fold_compound(aln)* – Initialize with a sequence alignment *aln* stored as a list of sequences (with gap characters)

The resulting object has a list of attached methods which in most cases directly correspond to functions that mainly operate on the corresponding *C* data structure:

- *type()* – Get the type of the *fold_compound* (See vrna_fc_type_e)
- *length()* – Get the length of the sequence(s) or alignment stored within the *fold_↩ compound*

Collaboration diagram for vrna_fc_s:

**Data Fields**

**Common data fields**

- vrna_fc_type_e type

    *The type of the vrna_fold_compound_t.*
- unsigned int length

    *The length of the sequence (or sequence alignment)*
- int cutpoint

    *The position of the (cofold) cutpoint within the provided sequence. If there is no cutpoint, this field will be set to -1.*
- unsigned int ∗ strand_number

    *The strand number a particular nucleotide is associated with.*
- unsigned int ∗ **strand_order**
- unsigned int ∗ **strand_start**
- unsigned int ∗ **strand_end**
- unsigned int **strands**
- vrna_seq_t ∗ **nucleotides**
- vrna_hc_t ∗ hc

    *The hard constraints data structure used for structure prediction.*
- vrna_mx_mfe_t ∗ matrices

    *The MFE DP matrices.*
- vrna_mx_pf_t ∗ exp_matrices

    *The PF DP matrices.*
- vrna_param_t ∗ params

    *The precomputed free energy contributions for each type of loop.*
- vrna_exp_param_t ∗ exp_params

    *The precomputed free energy contributions as Boltzmann factors.*
- int ∗ iindx

    *DP matrix accessor.*
- int ∗ jindx

*DP matrix accessor.*

## User-defined data fields

- vrna_callback_recursion_status ∗ stat_cb

  *Recursion status callback (usually called just before, and after recursive computations in the library.*
- void ∗ auxdata

  *A pointer to auxiliary, user-defined data.*
- vrna_callback_free_auxdata ∗ free_auxdata

  *A callback to free auxiliary user data whenever the fold_compound itself is free'd.*

## Secondary Structure Decomposition (grammar) related data fields

- vrna_sd_t ∗ domains_struc

  *Additional structured domains.*
- vrna_ud_t ∗ domains_up

  *Additional unstructured domains.*
- vrna_gr_aux_t ∗ **aux_grammar**

## Data fields available for single/hybrid structure prediction

## Data fields for consensus structure prediction

## Additional data fields for Distance Class Partitioning

*These data fields are typically populated with meaningful data only if used in the context of Distance Class Partitioning*

- unsigned int maxD1

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

  *Maximum allowed base pair distance to second reference.*
- short ∗ reference_pt1

  *A pairtable of the first reference structure.*
- short ∗ reference_pt2

  *A pairtable of the second reference structure.*
- 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.*

## Additional data fields for local folding

*These data fields are typically populated with meaningful data only if used in the context of local folding*

- int window_size

  *window size for local folding sliding window approach*
- char ∗∗ ptype_local

  *Pair type array (for local folding)*

**15.55.2.1.1   Field Documentation**

**15.55.2.1.1.1   type**

`vrna_fc_type_e vrna_fc_s::type`

The type of the vrna_fold_compound_t.

Currently possible values are VRNA_FC_TYPE_SINGLE, and VRNA_FC_TYPE_COMPARATIVE

**Warning**

> Do not edit this attribute, it will be automagically set by the corresponding get() methods for the vrna_fold_compound_t. The value specified in this attribute dictates the set of other attributes to use within this data structure.

**15.55.2.1.1.2   stat_cb**

`vrna_callback_recursion_status* vrna_fc_s::stat_cb`

Recursion status callback (usually called just before, and after recursive computations in the library.

**See also**

> vrna_callback_recursion_status(), vrna_fold_compound_add_callback()

**15.55.2.1.1.3   auxdata**

`void* vrna_fc_s::auxdata`

A pointer to auxiliary, user-defined data.

**See also**

> vrna_fold_compound_add_auxdata(), vrna_fold_compound_t.free_auxdata

**15.55.2.1.1.4   free_auxdata**

`vrna_callback_free_auxdata* vrna_fc_s::free_auxdata`

A callback to free auxiliary user data whenever the fold_compound itself is free'd.

**See also**

> vrna_fold_compound_t.auxdata, vrna_callback_free_auxdata()

**15.55.2.1.1.5 sequence**

```
char* vrna_fc_s::sequence
```

The input sequence string.

**Warning**

> Only available if
>
> ```
> type==VRNA_FC_TYPE_SINGLE
> ```

**15.55.2.1.1.6 sequence_encoding**

```
short* vrna_fc_s::sequence_encoding
```

Numerical encoding of the input sequence.

**See also**

> vrna_sequence_encode()

**Warning**

> Only available if
>
> ```
> type==VRNA_FC_TYPE_SINGLE
> ```

**15.55.2.1.1.7 ptype**

```
char* vrna_fc_s::ptype
```

Pair type array.

Contains the numerical encoding of the pair type for each pair (i,j) used in MFE, Partition function and Evaluation computations.

**Note**

> This array is always indexed via jindx, in contrast to previously different indexing between mfe and pf variants!

**Warning**

> Only available if
>
> ```
> type==VRNA_FC_TYPE_SINGLE
> ```

**See also**

> vrna_idx_col_wise(), vrna_ptypes()

**15.55.2.1.1.8 ptype_pf_compat**

```
char* vrna_fc_s::ptype_pf_compat
```

ptype array indexed via iindx

**Deprecated** This attribute will vanish in the future! It's meant for backward compatibility only!

**Warning**

> Only available if
>
> ```
> type==VRNA_FC_TYPE_SINGLE
> ```

**15.55.2.1.1.9 sc**

```
vrna_sc_t* vrna_fc_s::sc
```

The soft constraints for usage in structure prediction and evaluation.

**Warning**

> Only available if
>
> ```
> type==VRNA_FC_TYPE_SINGLE
> ```

**15.55.2.1.1.10 sequences**

```
char** vrna_fc_s::sequences
```

The aligned sequences.

**Note**

> The end of the alignment is indicated by a NULL pointer in the second dimension

**Warning**

> Only available if
>
> ```
> type==VRNA_FC_TYPE_COMPARATIVE
> ```

**15.55.2.1.1.11 n_seq**

```
unsigned int vrna_fc_s::n_seq
```

The number of sequences in the alignment.

**Warning**

> Only available if
>
> type==VRNA_FC_TYPE_COMPARATIVE

**15.55.2.1.1.12 cons_seq**

```
char* vrna_fc_s::cons_seq
```

The consensus sequence of the aligned sequences.

**Warning**

> Only available if
>
> type==VRNA_FC_TYPE_COMPARATIVE

**15.55.2.1.1.13 S_cons**

```
short* vrna_fc_s::S_cons
```

Numerical encoding of the consensus sequence.

**Warning**

> Only available if
>
> type==VRNA_FC_TYPE_COMPARATIVE

**15.55.2.1.1.14 S**

```
short** vrna_fc_s::S
```

Numerical encoding of the sequences in the alignment.

**Warning**

> Only available if
>
> type==VRNA_FC_TYPE_COMPARATIVE

**15.55.2.1.1.15 S5**

```
short** vrna_fc_s::S5
```

S5[s][i] holds next base 5' of i in sequence s.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.55.2.1.1.16 S3**

```
short** vrna_fc_s::S3
```

Sl[s][i] holds next base 3' of i in sequence s.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.55.2.1.1.17 pscore**

```
int* vrna_fc_s::pscore
```

Precomputed array of pair types expressed as pairing scores.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.55.2.1.1.18 pscore_local**

```
int** vrna_fc_s::pscore_local
```

Precomputed array of pair types expressed as pairing scores.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.55.2.1.1.19 pscore_pf_compat**

```
short* vrna_fc_s::pscore_pf_compat
```

Precomputed array of pair types expressed as pairing scores indexed via iindx.

**Deprecated** This attribute will vanish in the future!

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.55.2.1.1.20 scs**

```
vrna_sc_t** vrna_fc_s::scs
```

A set of soft constraints (for each sequence in the alignment)

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.55.3 Macro Definition Documentation**

**15.55.3.1 VRNA_STATUS_MFE_PRE**

```
#define VRNA_STATUS_MFE_PRE (unsigned char)1
```

#include <ViennaRNA/fold_compound.h>

Status message indicating that MFE computations are about to begin.

**See also**

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_mfe(), vrna_fold(), vrna_circfold(), vrna_alifold(), vrna_circalifold(), vrna_cofold()

### 15.55.3.2 VRNA_STATUS_MFE_POST

`#define VRNA_STATUS_MFE_POST (unsigned char)2`

`#include <`ViennaRNA/fold_compound.h`>`

Status message indicating that MFE computations are finished.

**See also**

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_mfe(), vrna_fold(), vrna_circfold(), vrna_alifold(), vrna_circalifold(), vrna_cofold()

### 15.55.3.3 VRNA_STATUS_PF_PRE

`#define VRNA_STATUS_PF_PRE (unsigned char)3`

`#include <`ViennaRNA/fold_compound.h`>`

Status message indicating that Partition function computations are about to begin.

**See also**

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_pf()

### 15.55.3.4 VRNA_STATUS_PF_POST

`#define VRNA_STATUS_PF_POST (unsigned char)4`

`#include <`ViennaRNA/fold_compound.h`>`

Status message indicating that Partition function computations are finished.

**See also**

vrna_fold_compound_t.stat_cb, vrna_callback_recursion_status(), vrna_pf()

### 15.55.3.5 VRNA_OPTION_MFE

`#define VRNA_OPTION_MFE 1U`

`#include <`ViennaRNA/fold_compound.h`>`

Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.

**See also**

vrna_fold_compound(), vrna_fold_compound_comparative(), VRNA_OPTION_EVAL_ONLY

### 15.55.3.6 VRNA_OPTION_PF

```
#define VRNA_OPTION_PF 2U
```

`#include <`ViennaRNA/fold_compound.h`>`

Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.

**See also**

> vrna_fold_compound(), vrna_fold_compound_comparative(), VRNA_OPTION_EVAL_ONLY

### 15.55.3.7 VRNA_OPTION_EVAL_ONLY

```
#define VRNA_OPTION_EVAL_ONLY 8U
```

`#include <`ViennaRNA/fold_compound.h`>`

Option flag to specify that neither MFE, nor PF DP matrices are required.

Use this flag in conjunction with VRNA_OPTION_MFE, and VRNA_OPTION_PF to save memory for a vrna_fold_compound_t obtained from vrna_fold_compound(), or vrna_fold_compound_comparative() in cases where only energy evaluation but no structure prediction is required.

**See also**

> vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_eval_structure()

## 15.55.4 Typedef Documentation

### 15.55.4.1 vrna_callback_free_auxdata

```
typedef void() vrna_callback_free_auxdata(void *data)
```

`#include <`ViennaRNA/fold_compound.h`>`

Callback to free memory allocated for auxiliary user-provided data.

This type of user-implemented function usually deletes auxiliary data structures. The user must take care to free all the memory occupied by the data structure passed.

**Notes on Callback Functions** This callback is supposed to free memory occupied by an auxiliary data structure. It will be called when the vrna_fold_compound_t is erased from memory through a call to vrna_fold_compound_free() and will be passed the address of memory previously bound to the vrna_fold_compound_t via vrna_fold_compound_add_auxdata().

**See also**

> vrna_fold_compound_add_auxdata(), vrna_fold_compound_free(), vrna_fold_compound_add_callback()

**Parameters**

| | |
|---|---|
| *data* | The data that needs to be free'd |

**15.55.4.2 vrna_callback_recursion_status**

```
typedef void() vrna_callback_recursion_status(unsigned char status, void *data)
```

`#include <ViennaRNA/fold_compound.h>`

Callback to perform specific user-defined actions before, or after recursive computations.

**Notes on Callback Functions** This function will be called to notify a third-party implementation about the status of a currently ongoing recursion. The purpose of this callback mechanism is to provide users with a simple way to ensure pre- and post conditions for auxiliary mechanisms attached to our implementations.

**See also**

vrna_fold_compound_add_auxdata(), vrna_fold_compound_add_callback(), vrna_mfe(), vrna_pf(), VRNA_STATUS_MFE_PRE, VRNA_STATUS_MFE_POST, VRNA_STATUS_PF_PRE, VRNA_STATUS_PF_POST

**Parameters**

| | |
|---|---|
| *status* | The status indicator |
| *data* | The data structure that was assigned with vrna_fold_compound_add_auxdata() |
| *status* | The status indicator |

**15.55.5 Enumeration Type Documentation**

**15.55.5.1 vrna_fc_type_e**

```
enum vrna_fc_type_e
```

`#include <ViennaRNA/fold_compound.h>`

An enumerator that is used to specify the type of a vrna_fold_compound_t.

**Enumerator**

| | |
|---|---|
| VRNA_FC_TYPE_SINGLE | Type is suitable for single, and hybridizing sequences |
| VRNA_FC_TYPE_COMPARATIVE | Type is suitable for sequence alignments (consensus structure prediction) |

## 15.55.6 Function Documentation

### 15.55.6.1 vrna_fold_compound()

vrna_fold_compound_t* vrna_fold_compound (
           const char * *sequence,*
           vrna_md_t * *md_p,*
           unsigned int *options* )

#include <ViennaRNA/fold_compound.h>

Retrieve a vrna_fold_compound_t data structure for single sequences and hybridizing sequences.

This function provides an easy interface to obtain a prefilled vrna_fold_compound_t by passing a single sequence, or two contatenated sequences as input. For the latter, sequences need to be seperated by an '&' character like this:

char *sequence = "GGGG&CCCC";

The optional parameter md_p can be used to specify the model details for successive computations based on the content of the generated vrna_fold_compound_t. Passing NULL will instruct the function to use default model details. The third parameter options may be used to specify dynamic programming (DP) matrix requirements. Use the macros:

- VRNA_OPTION_MFE

- VRNA_OPTION_PF

- VRNA_OPTION_WINDOW

- VRNA_OPTION_EVAL_ONLY

- VRNA_OPTION_DEFAULT

to specify the required type of computations that will be performed with the vrna_fold_compound_t.

If you just need the folding compound serving as a container for your data, you can simply pass VRNA_OPTION_DEFAULT to the option parameter. This creates a vrna_fold_compound_t without DP matrices, thus saving memory. Subsequent calls of any structure prediction function will then take care of allocating the memory required for the DP matrices. If you only intend to evaluate structures instead of actually predicting them, you may use the VRNA_OPTION_EVAL_ONLY macro. This will seriously speedup the creation of the vrna_fold_compound_t.

**Note**

> The sequence string must be uppercase, and should contain only RNA (resp. DNA) alphabet depending on what energy parameter set is used

**See also**

> vrna_fold_compound_free(), vrna_fold_compound_comparative(), vrna_md_t, VRNA_OPTION_MFE, VRNA_OPTION_PF, VRNA_OPTION_EVAL_ONLY, VRNA_OPTION_WINDOW

**Parameters**

| *sequence* | A single sequence, or two concatenated sequences seperated by an '&' character |
|---|---|
| *md_p* | An optional set of model details |
| *options* | The options for DP matrices memory allocation |

**Returns**

A prefilled vrna_fold_compound_t that can be readily used for computations

### 15.55.6.2 vrna_fold_compound_comparative()

vrna_fold_compound_t* vrna_fold_compound_comparative (
     const char ** *sequences,*
     vrna_md_t * *md_p,*
     unsigned int *options* )

#include <ViennaRNA/fold_compound.h>

Retrieve a vrna_fold_compound_t data structure for sequence alignments.

This function provides an easy interface to obtain a prefilled vrna_fold_compound_t by passing an alignment of sequences.

The optional parameter `md_p` can be used to specify the model details for successive computations based on the content of the generated vrna_fold_compound_t. Passing NULL will instruct the function to use default model details. The third parameter `options` may be used to specify dynamic programming (DP) matrix requirements. Use the macros:

- VRNA_OPTION_MFE

- VRNA_OPTION_PF

- VRNA_OPTION_EVAL_ONLY

- VRNA_OPTION_DEFAULT

to specify the required type of computations that will be performed with the vrna_fold_compound_t.

If you just need the folding compound serving as a container for your data, you can simply pass VRNA_OPTION_DEFAULT to the `option` parameter. This creates a vrna_fold_compound_t without DP matrices, thus saving memory. Subsequent calls of any structure prediction function will then take care of allocating the memory required for the DP matrices. If you only intend to evaluate structures instead of actually predicting them, you may use the VRNA_OPTION_EVAL_ONLY macro. This will seriously speedup the creation of the vrna_fold_compound_t.

**Note**

The sequence strings must be uppercase, and should contain only RNA (resp. DNA) alphabet including gap characters depending on what energy parameter set is used.

**See also**

vrna_fold_compound_free(), vrna_fold_compound(), vrna_md_t, VRNA_OPTION_MFE, VRNA_OPTION_PF, VRNA_OPTION_EVAL_ONLY, read_clustal()

**Parameters**

| | |
|---|---|
| *sequences* | A sequence alignment including 'gap' characters |
| *md_p* | An optional set of model details |
| *options* | The options for DP matrices memory allocation |

**Returns**

A prefilled vrna_fold_compound_t that can be readily used for computations

**15.55.6.3    vrna_fold_compound_free()**

```
void vrna_fold_compound_free (
            vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/fold_compound.h>
```

Free memory occupied by a vrna_fold_compound_t.

**See also**

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_mx_mfe_free(), vrna_mx_pf_free()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t that is to be erased from memory |

**15.55.6.4    vrna_fold_compound_add_auxdata()**

```
void vrna_fold_compound_add_auxdata (
            vrna_fold_compound_t * vc,
            void * data,
            vrna_callback_free_auxdata * f )
```

```
#include <ViennaRNA/fold_compound.h>
```

Add auxiliary data to the vrna_fold_compound_t.

This function allows one to bind arbitrary data to a vrna_fold_compound_t which may later on be used by one of the callback functions, e.g. vrna_callback_recursion_status(). To allow for proper cleanup of the memory occupied by this auxiliary data, the user may also provide a pointer to a cleanup function that free's the corresponding memory. This function will be called automatically when the vrna_fold_compound_t is free'd with vrna_fold_compound_free().

**Note**

Before attaching the arbitrary data pointer, this function will call the vrna_callback_free_auxdata() on any pre-existing data that is already attached.

**See also**

vrna_callback_free_auxdata()

**Parameters**

| vc | The fold_compound the arbitrary data pointer should be associated with |
|------|------------------------------------------------------------------------|
| data | A pointer to an arbitrary data structure |
| f | A pointer to function that free's memory occupied by the arbitrary data (May be NULL) |

**15.55.6.5 vrna_fold_compound_add_callback()**

```
void vrna_fold_compound_add_callback (
            vrna_fold_compound_t * vc,
            vrna_callback_recursion_status * f )
```

```
#include <ViennaRNA/fold_compound.h>
```

Add a recursion status callback to the vrna_fold_compound_t.

Binding a recursion status callback function to a vrna_fold_compound_t allows one to perform arbitrary operations just before, or after an actual recursive computations, e.g. MFE prediction, is performed by the RNAlib. The callback function will be provided with a pointer to its vrna_fold_compound_t, and a status message. Hence, it has complete access to all variables that incluence the recursive computations.

**See also**

vrna_callback_recursion_status(), vrna_fold_compound_t, VRNA_STATUS_MFE_PRE, VRNA_STATUS_MFE_POST, VRNA_STATUS_PF_PRE, VRNA_STATUS_PF_POST

**Parameters**

| vc | The fold_compound the callback function should be attached to |
|----|--------------------------------------------------------------|
| f | The pointer to the recursion status callback function |

## 15.56 The Dynamic Programming Matrices

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

### 15.56.1 Detailed Description

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

Collaboration diagram for The Dynamic Programming Matrices:

**Data Structures**

- struct vrna_mx_mfe_s

    *Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...*

- struct vrna_mx_pf_s

    *Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...*

**Typedefs**

- typedef struct vrna_mx_mfe_s vrna_mx_mfe_t

    *Typename for the Minimum Free Energy (MFE) DP matrices data structure vrna_mx_mfe_s.*

- typedef struct vrna_mx_pf_s vrna_mx_pf_t

    *Typename for the Partition Function (PF) DP matrices data structure vrna_mx_pf_s.*

**Enumerations**

- enum vrna_mx_type_e { VRNA_MX_DEFAULT, VRNA_MX_WINDOW, VRNA_MX_2DFOLD }

    *An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.*

**Functions**

- int vrna_mx_add (vrna_fold_compound_t ∗vc, vrna_mx_type_e type, unsigned int options)

    *Add Dynamic Programming (DP) matrices (allocate memory)*

- void vrna_mx_mfe_free (vrna_fold_compound_t ∗vc)

    *Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.*

- void vrna_mx_pf_free (vrna_fold_compound_t ∗vc)

    *Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.*

## 15.56.2 Data Structure Documentation

### 15.56.2.1 struct vrna_mx_mfe_s

Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t.

**Data Fields**

#### Common fields for MFE matrices

- vrna_mx_type_e **type**
- unsigned int length

  *Length of the sequence, therefore an indicator of the size of the DP matrices.*

#### Default DP matrices

*Note*

  *These data fields are available if*

  vrna_mx_mfe_t.type == VRNA_MX_DEFAULT

#### Local Folding DP matrices using window approach

*Note*

  *These data fields are available if*

  vrna_mx_mfe_t.type == VRNA_MX_WINDOW

#### Distance Class DP matrices

*Note*

  *These data fields are available if*

  vrna_mx_mfe_t.type == VRNA_MX_2DFOLD

### 15.56.2.2 struct vrna_mx_pf_s

Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t.

**Data Fields**

### Common fields for DP matrices

- vrna_mx_type_e **type**
- unsigned int **length**
- FLT_OR_DBL ∗ **scale**
- FLT_OR_DBL ∗ **expMLbase**

### Default PF matrices

*Note*

> *These data fields are available if*
>
> vrna_mx_pf_t.type == VRNA_MX_DEFAULT

### Local Folding DP matrices using window approach

*Note*

> *These data fields are available if*
>
> vrna_mx_mfe_t.type == VRNA_MX_WINDOW

### Distance Class DP matrices

*Note*

> *These data fields are available if*
>
> vrna_mx_pf_t.type == VRNA_MX_2DFOLD

## 15.56.3 Enumeration Type Documentation

### 15.56.3.1 vrna_mx_type_e

enum vrna_mx_type_e

#include <ViennaRNA/dp_matrices.h>

An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.

**See also**

> vrna_mx_mfe_t, vrna_mx_pf_t

**Enumerator**

| VRNA_MX_DEFAULT | Default DP matrices. |
|---|---|
| VRNA_MX_WINDOW | DP matrices suitable for local structure prediction using window approach. **See also** vrna_mfe_window(), vrna_mfe_window_zscore(), pfl_fold() |
| VRNA_MX_2DFOLD | DP matrices suitable for distance class partitioned structure prediction. **See also** vrna_mfe_TwoD(), vrna_pf_TwoD() |

### 15.56.4  Function Documentation

#### 15.56.4.1  vrna_mx_add()

```
int vrna_mx_add (
            vrna_fold_compound_t * vc,
            vrna_mx_type_e type,
            unsigned int options )
```

`#include <ViennaRNA/dp_matrices.h>`

Add Dynamic Programming (DP) matrices (allocate memory)

This function adds DP matrices of a specific type to the provided vrna_fold_compound_t, such that successive DP recursion can be applied. The function caller has to specify which type of DP matrix is requested, see vrna_mx_type_e, and what kind of recursive algorithm will be applied later on, using the parameters type, and options, respectively. For the latter, Minimum free energy (MFE), and Partition function (PF) computations are distinguished. A third option that may be passed is VRNA_OPTION_HYBRID, indicating that auxiliary DP arrays are required for RNA-RNA interaction prediction.

**Note**

> Usually, there is no need to call this function, since the constructors of vrna_fold_compound_t are handling all the DP matrix memory allocation.

**See also**

> vrna_mx_mfe_add(), vrna_mx_pf_add(), vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), vrna_mx_pf_free(), vrna_mx_mfe_free(), vrna_mx_type_e, VRNA_OPTION_MFE, VRNA_OPTION_PF, VRNA_OPTION_HYBRID, VRNA_OPTION_EVAL_ONLY

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t that holds pointers to the DP matrices |
| *type* | The type of DP matrices requested |
| *options* | Option flags that specify the kind of DP matrices, such as MFE or PF arrays, and auxiliary requirements |

**Returns**

> 1 if DP matrices were properly allocated and attached, 0 otherwise

#### 15.56.4.2  vrna_mx_mfe_free()

```
void vrna_mx_mfe_free (
            vrna_fold_compound_t * vc )
```

`#include <ViennaRNA/dp_matrices.h>`

Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.

**See also**

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), vrna_mx_pf_free()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t storing the MFE DP matrices that are to be erased from memory |

**15.56.4.3   vrna_mx_pf_free()**

```
void vrna_mx_pf_free (
            vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/dp_matrices.h>
```

Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

**See also**

vrna_fold_compound(), vrna_fold_compound_comparative(), vrna_fold_compound_free(), vrna_mx_mfe_free()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t storing the PF DP matrices that are to be erased from memory |

## 15.57 Direct refolding paths between two secondary structures

Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.

### 15.57.1 Detailed Description

Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.

Collaboration diagram for Direct refolding paths between two secondary structures:

**Data Structures**

- struct vrna_path_s

  *An element of a refolding path list. More...*

**Typedefs**

- typedef struct vrna_path_s vrna_path_t

  *Typename for the refolding path data structure vrna_path_s.*

- typedef struct vrna_path_s path_t

  *Old typename of vrna_path_s.*

**Functions**

- int vrna_path_findpath_saddle (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width)

  *Find energy of a saddle point between 2 structures (search only direct path)*

- int vrna_path_findpath_saddle_ub (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width, int maxE)

  *Find energy of a saddle point between 2 structures (search only direct path)*

- vrna_path_t ∗ vrna_path_findpath (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width)

  *Find refolding path between 2 structures (search only direct path)*

- vrna_path_t ∗ vrna_path_findpath_ub (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width, int maxE)

  *Find refolding path between 2 structures (search only direct path)*

- int find_saddle (const char ∗seq, const char ∗s1, const char ∗s2, int width)

  *Find energy of a saddle point between 2 structures (search only direct path)*

- void free_path (vrna_path_t ∗path)

  *Free memory allocated by get_path() function.*

- vrna_path_t ∗ get_path (const char ∗seq, const char ∗s1, const char ∗s2, int width)

  *Find refolding path between 2 structures (search only direct path)*

## 15.57.2 Data Structure Documentation

### 15.57.2.1 struct vrna_path_s

An element of a refolding path list.

**See also**

> vrna_path_findpath()

**Data Fields**

- double en
    - *Free energy of current structure.*
- char ∗ s
    - *Secondary structure in dot-bracket notation.*

## 15.57.3 Typedef Documentation

### 15.57.3.1 path_t

```
typedef struct vrna_path_s path_t
```

```
#include <ViennaRNA/findpath.h>
```

Old typename of vrna_path_s.

**Deprecated** Use vrna_path_t instead!

## 15.57.4 Function Documentation

### 15.57.4.1 vrna_path_findpath_saddle()

```
vrna_path_findpath_saddle (
            vrna_fold_compound_t * vc,
            const char * s1,
            const char * s2,
            int width )
```

```
#include <ViennaRNA/findpath.h>
```

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

This function uses an inplementation of the *findpath* algorithm [5] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

**See also**

> vrna_path_findpath_saddle_ub(), vrna_fold_compound(), vrna_fold_compound_t, vrna_path_findpath()

**Parameters**

| vc | The vrna_fold_compound_t with precomputed sequence encoding and model details |
|---|---|
| s1 | The start structure in dot-bracket notation |
| s2 | The target structure in dot-bracket notation |
| width | A number specifying how many strutures are being kept at each step during the search |

**Returns**

> The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path_findpath_saddle()* to objects of type *fold_compound*. The optional parameter `width` defaults to 1 if it is omitted.

**15.57.4.2 vrna_path_findpath_saddle_ub()**

```
vrna_path_findpath_saddle_ub (
            vrna_fold_compound_t * vc,
            const char * s1,
            const char * s2,
            int width,
            int maxE )
```

#include <ViennaRNA/findpath.h>

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

This function uses an inplementation of the *findpath* algorithm [5] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

**Warning**

> The argument `maxE` ( $E_{max}$) enables one to specify an upper bound, or maximum free energy for the saddle point between the two input structures. If no path with $E_{saddle} < E_{max}$ is found, the function simply returns `maxE`

**See also**

> vrna_path_findpath_saddle(), vrna_fold_compound(), vrna_fold_compound_t, vrna_path_findpath()

**Parameters**

| vc | The vrna_fold_compound_t with precomputed sequence encoding and model details |
|---|---|
| s1 | The start structure in dot-bracket notation |
| s2 | The target structure in dot-bracket notation |
| width | A number specifying how many strutures are being kept at each step during the search |
| maxE | An upper bound for the saddle point energy in 10cal/mol |

**Returns**

> The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path_findpath_saddle()* to objects of type *fold_compound*. The optional parameter `width` defaults to 1 if it is omitted, while the optional parameter `maxE` defaults to INF. In case the function did not find a path with $E_{saddle} < E_{max}$ the function returns a *NULL* object, i.e. *undef* for Perl and *None* for Python.

**15.57.4.3   vrna_path_findpath()**

```
vrna_path_findpath (
            vrna_fold_compound_t * vc,
            const char * s1,
            const char * s2,
            int width )
```

`#include <ViennaRNA/findpath.h>`

Find refolding path between 2 structures (search only direct path)

This function uses an inplementation of the *findpath* algorithm [5] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

**See also**

> vrna_path_findpath_ub(), vrna_fold_compound(), vrna_fold_compound_t, vrna_path_findpath_saddle()

**Parameters**

| vc | The vrna_fold_compound_t with precomputed sequence encoding and model details |
|---|---|
| s1 | The start structure in dot-bracket notation |
| s2 | The target structure in dot-bracket notation |
| width | A number specifying how many strutures are being kept at each step during the search |

**Returns**

> The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path_findpath()* to objects of type *fold←_compound*. The optional parameter `width` defaults to 1 if it is omitted.

**15.57.4.4  vrna_path_findpath_ub()**

```
vrna_path_findpath_ub (
            vrna_fold_compound_t * vc,
            const char * s1,
            const char * s2,
            int width,
            int maxE )
```

`#include <ViennaRNA/findpath.h>`

Find refolding path between 2 structures (search only direct path)

This function uses an inplementation of the *findpath* algorithm [5] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The vrna_fold_compound_t does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

**Warning**

The argument `maxE` enables one to specify an upper bound, or maximum free energy for the saddle point between the two input structures. If no path with $E_{saddle} < E_{max}$ is found, the function simply returns *NULL*

**See also**

vrna_path_findpath(), vrna_fold_compound(), vrna_fold_compound_t, vrna_path_findpath_saddle()

**Parameters**

| | |
|---|---|
| *vc* | The vrna_fold_compound_t with precomputed sequence encoding and model details |
| *s1* | The start structure in dot-bracket notation |
| *s2* | The target structure in dot-bracket notation |
| *width* | A number specifying how many strutures are being kept at each step during the search |
| *maxE* | An upper bound for the saddle point energy in 10cal/mol |

**Returns**

The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path_findpath()* to objects of type *fold↩ _compound*. The optional parameter `width` defaults to 1 if it is omitted, while the optional parameter `maxE` defaults to INF. In case the function did not find a path with $E_{saddle} < E_{max}$ the function returns an empty list.

**15.57.4.5 find_saddle()**

```
int find_saddle (
            const char * seq,
            const char * s1,
            const char * s2,
            int width )
```

#include <ViennaRNA/findpath.h>

Find energy of a saddle point between 2 structures (search 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 |
| width | integer how many strutures are being kept during the search |

**Returns**

the saddle energy in 10cal/mol

**15.57.4.6 free_path()**

```
void free_path (
            vrna_path_t * path )
```

#include <ViennaRNA/findpath.h>

Free memory allocated by get_path() function.

**Parameters**

| path | pointer to memory to be freed |
|---|---|

**15.57.4.7 get_path()**

```
vrna_path_t* get_path (
            const char * seq,
            const char * s1,
            const char * s2,
            int width )
```

```
#include <ViennaRNA/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

```
#include <ViennaRNA/findpath.h>
```

**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 |
| *width* | integer how many strutures are being kept during the search |

**Returns**

direct refolding path between two structures

## 15.58 Parsing, converting, and comparing sequences

### 15.58.1 Detailed Description

Collaboration diagram for Parsing, converting, and comparing sequences:

### Macros

- #define XSTR(s) STR(s)

  *Stringify a macro after expansion.*
- #define STR(s) #s

  *Stringify a macro argument.*
- #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

- char ∗ vrna_strdup_printf (const char ∗format,...)

  *Safely create a formatted string.*
- char ∗ vrna_strdup_vprintf (const char ∗format, va_list argp)

  *Safely create a formatted string.*
- int vrna_strcat_printf (char ∗∗dest, const char ∗format,...)

  *Safely append a formatted string to another string.*
- int vrna_strcat_vprintf (char ∗∗dest, const char ∗format, va_list args)

  *Safely append a formatted string to another string.*
- char ∗∗ vrna_strsplit (const char ∗string, const char ∗delimiter)

  *Split a string into tokens using a delimiting character.*
- char ∗ vrna_random_string (int l, const char symbols[ ])

  *Create a random string using characters from a specified symbol set.*
- int vrna_hamming_distance (const char ∗s1, const char ∗s2)

  *Calculate hamming distance between two sequences.*
- int vrna_hamming_distance_bound (const char ∗s1, const char ∗s2, int n)

  *Calculate hamming distance between two sequences up to a specified length.*
- void vrna_seq_toRNA (char ∗sequence)

  *Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.*
- void vrna_seq_toupper (char ∗sequence)

  *Convert an input sequence to uppercase.*
- char ∗ vrna_cut_point_insert (const char ∗string, int cp)

  *Add a separating '&' character into a string according to cut-point position.*
- char ∗ vrna_cut_point_remove (const char ∗string, int ∗cp)

  *Remove a separating '&' character from a string.*

### 15.58.2 Macro Definition Documentation

**15.58.2.1 FILENAME_MAX_LENGTH**

```
#define FILENAME_MAX_LENGTH 80
```

#include <ViennaRNA/string_utils.h>

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.

**15.58.2.2 FILENAME_ID_LENGTH**

```
#define FILENAME_ID_LENGTH 42
```

#include <ViennaRNA/string_utils.h>

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

## 15.58.3 Function Documentation

**15.58.3.1 vrna_strdup_printf()**

```
char* vrna_strdup_printf (
            const char * format,
             ... )
```

#include <ViennaRNA/string_utils.h>

Safely create a formatted string.

This function is a safe implementation for creating a formatted character array, similar to *sprintf*. Internally, it uses the *asprintf* function if available to dynamically allocate a large enough character array to store the supplied content. If *asprintf* is not available, mimic it's behavior using *vsnprintf*.

**Note**

> The returned pointer of this function should always be passed to *free()* to release the allocated memory

**See also**

> vrna_strdup_vprintf(), vrna_strcat_printf()

**Parameters**

| *format* | The format string (See also asprintf) |
|----------|----------------------------------------|
| *...* | The list of variables used to fill the format string |

**Returns**

The formatted, null-terminated string, or NULL if something has gone wrong

**15.58.3.2  vrna_strdup_vprintf()**

```
char* vrna_strdup_vprintf (
            const char * format,
            va_list argp )
```

`#include <`ViennaRNA/string_utils.h`>`

Safely create a formatted string.

This function is the *va_list* version of vrna_strdup_printf()

**Note**

The returned pointer of this function should always be passed to *free()* to release the allocated memory

**See also**

vrna_strdup_printf(), vrna_strcat_printf(), vrna_strcat_vprintf()

**Parameters**

| *format* | The format string (See also asprintf) |
|----------|----------------------------------------|
| *...* | The list of variables used to fill the format string |

**Returns**

The formatted, null-terminated string, or NULL if something has gone wrong

**15.58.3.3  vrna_strcat_printf()**

```
int vrna_strcat_printf (
            char ** dest,
            const char * format,
             ... )
```

```
#include <ViennaRNA/string_utils.h>
```

Safely append a formatted string to another string.

This function is a safe implementation for appending a formatted character array, similar to a cobination of *strcat* and *sprintf*. The function automatically allocates enough memory to store both, the previous content stored at `dest` and the appended format string. If the `dest` pointer is NULL, the function allocate memory only for the format string. The function returns the number of characters in the resulting string or -1 in case of an error.

**See also**

> vrna_strcat_vprintf(), vrna_strdup_printf(), vrna_strdup_vprintf()

**Parameters**

| | |
|---|---|
| *dest* | The address of a char *pointer where the formatted string is to be appended |
| *format* | The format string (See also sprintf) |
| *...* | The list of variables used to fill the format string |

**Returns**

> The number of characters in the final string, or -1 on error

**15.58.3.4 vrna_strcat_vprintf()**

```
int vrna_strcat_vprintf (
        char ** dest,
        const char * format,
        va_list args )
```

```
#include <ViennaRNA/string_utils.h>
```

Safely append a formatted string to another string.

This function is the *va_list* version of vrna_strcat_printf()

**See also**

> vrna_strcat_printf(), vrna_strdup_printf(), vrna_strdup_vprintf()

**Parameters**

| | |
|---|---|
| *dest* | The address of a char *pointer where the formatted string is to be appended |
| *format* | The format string (See also sprintf) |
| *...* | The list of variables used to fill the format string |

**Returns**

The number of characters in the final string, or -1 on error

**15.58.3.5 vrna_strsplit()**

```
char** vrna_strsplit (
            const char * string,
            const char * delimiter )
```

```
#include <ViennaRNA/string_utils.h>
```

Split a string into tokens using a delimiting character.

This function splits a string into an array of strings using a single character that delimits the elements within the string. The default delimiter is the ampersand '&' and will be used when NULL is passed as a second argument. The returned list is NULL terminated, i.e. the last element is NULL. If the delimiter is not found, the returned list contains exactly one element: the input string.

For instance, the following code:

```
char **tok = vrna_strsplit("GGGG&CCCC&AAAAA", NULL);

for (char **ptr = tok; *ptr; ptr++) {
 printf("%s\n", *ptr);
 free(*ptr);
}
free(tok);
```

produces this output:

```
GGGG
CCCC
AAAAA
```

and properly free's the memory occupied by the returned element array.

**Note**

This function internally uses *strtok_r()* and is therefore considered to be thread-safe. Also note, that it is the users responsibility to free the memory of the array and that of the individual element strings!

**Parameters**

| | |
|---|---|
| *string* | The input string that should be split into elements |
| *delimiter* | The delimiting character. If NULL, the delimiter is "&" |

**Returns**

A NULL terminated list of the elements in the string

**15.58.3.6 vrna_random_string()**

```
char* vrna_random_string (
            int l,
            const char symbols[] )
```

#include <ViennaRNA/string_utils.h>

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

**15.58.3.7 vrna_hamming_distance()**

```
int vrna_hamming_distance (
            const char * s1,
            const char * s2 )
```

#include <ViennaRNA/string_utils.h>

Calculate hamming distance between two sequences.

**Parameters**

| | |
|---|---|
| *s1* | The first sequence |
| *s2* | The second sequence |

**Returns**

The hamming distance between s1 and s2

**15.58.3.8 vrna_hamming_distance_bound()**

```
int vrna_hamming_distance_bound (
            const char * s1,
            const char * s2,
            int n )
```

#include <ViennaRNA/string_utils.h>

```
#include <ViennaRNA/string_utils.h>
```

Calculate hamming distance between two sequences up to a specified length.

This function is similar to vrna_hamming_distance() 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 |
| *n* | The length of the subsequences to consider (starting from the 5' end) |

**Returns**

The hamming distance between s1 and s2

**15.58.3.9  vrna_seq_toRNA()**

```
void vrna_seq_toRNA (
            char * sequence )
```

```
#include <ViennaRNA/string_utils.h>
```

Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.

This function substitudes *T* and *t* with *U* and *u*, respectively

**Parameters**

| *sequence* | The sequence to be converted |
|------------|------------------------------|

**15.58.3.10  vrna_seq_toupper()**

```
void vrna_seq_toupper (
            char * sequence )
```

```
#include <ViennaRNA/string_utils.h>
```

Convert an input sequence to uppercase.

**Parameters**

| *sequence* | The sequence to be converted |
|------------|------------------------------|

**15.58.3.11 vrna_cut_point_insert()**

```
char* vrna_cut_point_insert (
            const char * string,
            int cp )
```

#include <ViennaRNA/string_utils.h>

Add a separating '&' character into a string according to cut-point position.

If the cut-point position is less or equal to zero, this function just returns a copy of the provided string. Otherwise, the cut-point character is set at the corresponding position

**Parameters**

| | |
|---|---|
| *string* | The original string |
| *cp* | The cut-point position |

**Returns**

A copy of the provided string including the cut-point character

**15.58.3.12 vrna_cut_point_remove()**

```
char* vrna_cut_point_remove (
            const char * string,
            int * cp )
```

#include <ViennaRNA/string_utils.h>

Remove a separating '&' character from a string.

This function removes the cut-point indicating '&' character from a string and memorizes its position in a provided integer variable. If not '&' is found in the input, the integer variable is set to -1. The function returns a copy of the input string with the '&' being sliced out.

**Parameters**

| | |
|---|---|
| *string* | The original string |
| *cp* | The cut-point position |

**Returns**

A copy of the input string with the '&' being sliced out

## 15.59 Parsing, converting, comparing secondary structures

### 15.59.1 Detailed Description

Collaboration diagram for Parsing, converting, comparing secondary structures:

### Files

- file RNAstruct.h

  *Parsing and Coarse Graining of Structures.*

### Data Structures

- struct vrna_hx_s

  *Data structure representing an entry of a helix list. More...*
- struct vrna_elem_prob_s

  *Data structure representing a single entry of an element probability list (e.g. list of pair probabilities) More...*

### Macros

- #define VRNA_BRACKETS_ALPHA 4U

  *Bitflag to indicate secondary structure notations using uppercase/lowercase letters from the latin alphabet.*
- #define VRNA_BRACKETS_RND 8U

  *Bitflag to indicate secondary structure notations using round brackets (parenthesis), ()*
- #define VRNA_BRACKETS_CLY 16U

  *Bitflag to indicate secondary structure notations using curly brackets, {}*
- #define VRNA_BRACKETS_ANG 32U

  *Bitflag to indicate secondary structure notations using angular brackets, <>*
- #define VRNA_BRACKETS_SQR 64U

  *Bitflag to indicate secondary structure notations using square brackets, []*
- #define VRNA_BRACKETS_DEFAULT

  *Default bitmask to indicate secondary structure notation using any pair of brackets.*

### Typedefs

- typedef struct vrna_hx_s vrna_hx_t

  *Convenience typedef for data structure vrna_hx_s.*
- typedef struct vrna_elem_prob_s vrna_ep_t

  *Convenience typedef for data structure #vrna_pp_s.*

**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.*

- char * vrna_db_pack (const char ∗struc)

  *Pack secondary secondary structure, 5:1 compression using base 3 encoding.*

- char * vrna_db_unpack (const char ∗packed)

  *Unpack secondary structure previously packed with vrna_db_pack()*

- short * vrna_ptable (const char ∗structure)

  *Create a pair table of a secondary structure.*

- short * vrna_ptable_from_string (const char ∗string, unsigned int options)

  *Create a pair table for a secondary structure string.*

- short * vrna_pt_pk_get (const char ∗structure)

  *Create a pair table of a secondary structure (pseudo-knot version)*

- short * vrna_ptable_copy (const short ∗pt)

  *Get an exact copy of a pair table.*

- short * vrna_pt_ali_get (const char ∗structure)

  *Create a pair table of a secondary structure (snoop align version)*

- short * vrna_pt_snoop_get (const char ∗structure)

  *Create a pair table of a secondary structure (snoop version)*

- int * vrna_loopidx_from_ptable (const short ∗pt)

  *Get a loop index representation of a structure.*

- void vrna_db_flatten (char ∗structure, unsigned int options)

  *Substitute pairs of brackets in a string with parenthesis.*

- void vrna_db_flatten_to (char ∗string, const char target[3], unsigned int options)

  *Substitute pairs of brackets in a string with another type of pair characters.*

- char * vrna_db_from_ptable (short ∗pt)

  *Convert a pair table into dot-parenthesis notation.*

- char * vrna_db_from_WUSS (const char ∗wuss)

  *Convert a WUSS annotation string to dot-bracket format.*

- int vrna_bp_distance (const char ∗str1, const char ∗str2)

*Compute the "base pair" distance between two secondary structures s1 and s2.*

• unsigned int ∗ vrna_refBPcnt_matrix (const short ∗reference_pt, unsigned int turn)

*Make a reference base pair count matrix.*

• unsigned int ∗ vrna_refBPdist_matrix (const short ∗pt1, const short ∗pt2, unsigned int turn)

*Make a reference base pair distance matrix.*

• char ∗ vrna_db_from_probs (const FLT_OR_DBL ∗pr, unsigned int length)

*Create a dot-bracket like structure string from base pair probability matrix.*

• char vrna_bpp_symbol (const float ∗x)

*Get a pseudo dot bracket notation for a given probability information.*

• char ∗ vrna_db_from_bp_stack (vrna_bp_stack_t ∗bp, unsigned int length)

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

• vrna_ep_t ∗ vrna_plist (const char ∗struc, float pr)

*Create a vrna_ep_t from a dot-bracket string.*

• char ∗ vrna_db_from_plist (vrna_ep_t ∗pairs, unsigned int n)

*Convert a list of base pairs into dot-bracket notation.*

• void assign_plist_from_db (vrna_ep_t ∗∗pl, const char ∗struc, float pr)

*Create a vrna_ep_t from a dot-bracket string.*

• 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 bp_distance (const char ∗str1, const char ∗str2)

*Compute the "base pair" distance between two secondary structures s1 and s2.*

• unsigned int ∗ make_referenceBP_array (short ∗reference_pt, unsigned int turn)

*Make a reference base pair count matrix.*

• unsigned int ∗ compute_BPdifferences (short ∗pt1, short ∗pt2, unsigned int turn)

*Make a reference base pair distance matrix.*

• void parenthesis_structure (char ∗structure, vrna_bp_stack_t ∗bp, int length)

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

• void parenthesis_zuker (char ∗structure, vrna_bp_stack_t ∗bp, int length)

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

• void bppm_to_structure (char ∗structure, FLT_OR_DBL ∗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.*

## 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.*

## 15.59.2 Data Structure Documentation

### 15.59.2.1 struct vrna_hx_s

Data structure representing an entry of a helix list.

### 15.59.2.2 struct vrna_elem_prob_s

Data structure representing a single entry of an element probability list (e.g. list of pair probabilities)

**See also**

vrna_plist(), vrna_plist_from_probs(), vrna_db_from_plist(), #VRNA_PLIST_TYPE_BASEPAIR, #VRNA_P↩
LIST_TYPE_GQUAD, #VRNA_PLIST_TYPE_H_MOTIF, #VRNA_PLIST_TYPE_I_MOTIF, #VRNA_PLIST_↩
TYPE_UD_MOTIF, #VRNA_PLIST_TYPE_STACK

## 15.59.3 Macro Definition Documentation

### 15.59.3.1 VRNA_BRACKETS_ALPHA

```
#define VRNA_BRACKETS_ALPHA 4U
```

```
#include <ViennaRNA/structure_utils.h>
```

Bitflag to indicate secondary structure notations using uppercase/lowercase letters from the latin alphabet.

**See also**

vrna_ptable_from_string()

**15.59.3.2 VRNA_BRACKETS_RND**

`#define VRNA_BRACKETS_RND 8U`

`#include <`[`ViennaRNA/structure_utils.h`]`>`

Bitflag to indicate secondary structure notations using round brackets (parenthesis), `()`

**See also**

> [vrna_ptable_from_string()](), [vrna_db_flatten()](), [vrna_db_flatten_to()]()

**15.59.3.3 VRNA_BRACKETS_CLY**

`#define VRNA_BRACKETS_CLY 16U`

`#include <`[`ViennaRNA/structure_utils.h`]`>`

Bitflag to indicate secondary structure notations using curly brackets, `{}`

**See also**

> [vrna_ptable_from_string()](), [vrna_db_flatten()](), [vrna_db_flatten_to()]()

**15.59.3.4 VRNA_BRACKETS_ANG**

`#define VRNA_BRACKETS_ANG 32U`

`#include <`[`ViennaRNA/structure_utils.h`]`>`

Bitflag to indicate secondary structure notations using angular brackets, $<>$

**See also**

> [vrna_ptable_from_string()](), [vrna_db_flatten()](), [vrna_db_flatten_to()]()

**15.59.3.5 VRNA_BRACKETS_SQR**

`#define VRNA_BRACKETS_SQR 64U`

`#include <`[`ViennaRNA/structure_utils.h`]`>`

Bitflag to indicate secondary structure notations using square brackets, `[]`

**See also**

> [vrna_ptable_from_string()](), [vrna_db_flatten()](), [vrna_db_flatten_to()]()

**15.59.3.6 VRNA_BRACKETS_DEFAULT**

```
#define VRNA_BRACKETS_DEFAULT
```

#include <[ViennaRNA/structure_utils.h](#)>

**Value:**

```
(VRNA_BRACKETS_RND | \
    VRNA_BRACKETS_CLY | \
    VRNA_BRACKETS_ANG | \
    VRNA_BRACKETS_SQR)
```

Default bitmask to indicate secondary structure notation using any pair of brackets.

**See also**

> vrna_ptable_from_string(), vrna_db_flatten(), vrna_db_flatten_to()

## 15.59.4 Function Documentation

**15.59.4.1 b2HIT()**

```
char* b2HIT (
            const char * structure )
```

#include <[ViennaRNA/RNAstruct.h](#)>

Converts the full structure from bracket notation to the HIT notation including root.

**Parameters**

| structure | |
| --- | --- |

**Returns**

**15.59.4.2 b2C()**

```
char* b2C (
            const char * structure )
```

#include <[ViennaRNA/RNAstruct.h](#)>

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**

### 15.59.4.3 b2Shapiro()

```
char* b2Shapiro (
            const char * structure )
```

`#include <`ViennaRNA/RNAstruct.h`>`

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**

### 15.59.4.4 add_root()

```
char* add_root (
            const char * structure )
```

`#include <`ViennaRNA/RNAstruct.h`>`

Adds a root to an un-rooted tree in any except bracket notation.

**Parameters**

| | |
|---|---|
| *structure* | |

**Returns**

**15.59.4.5 expand_Shapiro()**

```
char* expand_Shapiro (
            const char * coarse )
```

#include <[ViennaRNA/RNAstruct.h](ViennaRNA/RNAstruct.h)>

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from [b2C()](b2C()).

**Parameters**

| *coarse* | |
|---|---|

**Returns**

**15.59.4.6 expand_Full()**

```
char* expand_Full (
            const char * structure )
```

#include <[ViennaRNA/RNAstruct.h](ViennaRNA/RNAstruct.h)>

Convert the full structure from bracket notation to the expanded notation including root.

**Parameters**

| *structure* | |
|---|---|

**Returns**

**15.59.4.7 unexpand_Full()**

```
char* unexpand_Full (
            const char * ffull )
```

#include <[ViennaRNA/RNAstruct.h](ViennaRNA/RNAstruct.h)>

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**

**15.59.4.8 unweight()**

```
char* unweight (
            const char * wcoarse )
```

#include <ViennaRNA/RNAstruct.h>

Strip weights from any weighted tree.

**Parameters**

| wcoarse | |
|---------|--|

**Returns**

**15.59.4.9 unexpand_aligned_F()**

```
void unexpand_aligned_F (
            char * align[2] )
```

#include <ViennaRNA/RNAstruct.h>

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 | |
|-------|--|

**15.59.4.10 parse_structure()**

```
void parse_structure (
            const char * structure )
```

#include <ViennaRNA/RNAstruct.h>

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**

**15.59.4.11    vrna_db_pack()**

```
char* vrna_db_pack (
            const char * struc )
```

#include <[ViennaRNA/structure_utils.h](ViennaRNA/structure_utils.h)>

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.

**See also**

[vrna_db_unpack()](vrna_db_unpack)

**Parameters**

| *struc* | The secondary structure in dot-bracket notation |
|---|---|

**Returns**

The binary encoded structure

**15.59.4.12    vrna_db_unpack()**

```
char* vrna_db_unpack (
            const char * packed )
```

#include <[ViennaRNA/structure_utils.h](ViennaRNA/structure_utils.h)>

Unpack secondary structure previously packed with [vrna_db_pack()](vrna_db_pack)

Translate a compressed binary string produced by [vrna_db_pack()](vrna_db_pack) back into the familiar dot-bracket notation.

**See also**

[vrna_db_pack()](vrna_db_pack)

**Parameters**

| *packed* | The binary encoded packed secondary structure |
|---|---|

**Returns**

The unpacked secondary structure in dot-bracket notation

**15.59.4.13 vrna_ptable()**

```
short* vrna_ptable (
            const char * structure )
```

#include <ViennaRNA/structure_utils.h>

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.

**See also**

vrna_ptable_from_string(), vrna_db_from_ptable()

**Parameters**

| *structure* | The secondary structure in dot-bracket notation |
|---|---|

**Returns**

A pointer to the created pair_table

**15.59.4.14 vrna_ptable_from_string()**

```
short* vrna_ptable_from_string (
            const char * string,
            unsigned int options )
```

#include <ViennaRNA/structure_utils.h>

Create a pair table for a secondary structure string.

This function takes an input string of a secondary structure annotation in Dot-Bracket Notation (a.k.a. Dot-Parenthesis Notation) or Extended Dot-Bracket Notation, and converts it into a pair table representation.

**See also**

vrna_ptable(), vrna_db_from_ptable(), vrna_db_flatten_to(), VRNA_BRACKETS_RND, VRNA_BRACKETS_ANG, VRNA_BRACKETS_CLY, VRNA_BRACKETS_SQR, VRNA_BRACKETS_DEFAULT

**Parameters**

| | |
|---|---|
| *string* | Secondary structure in Extended Dot-Bracket Notation |
| *options* | A bitmask to specify which brackets are recognized during conversion to pair table |

**Returns**

A pointer to a new pair table of the provided secondary structure

**15.59.4.15   vrna_pt_pk_get()**

```
short* vrna_pt_pk_get (
            const char * structure )
```

#include <ViennaRNA/structure_utils.h>

Create a pair table of a secondary structure (pseudo-knot version)

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.

In contrast to vrna_ptable() this function also recognizes the base pairs denoted by '[' and ']' brackets.

**Parameters**

| | |
|---|---|
| *structure* | The secondary structure in (extended) dot-bracket notation |

**Returns**

A pointer to the created pair_table

**15.59.4.16   vrna_ptable_copy()**

```
short* vrna_ptable_copy (
            const short * pt )
```

#include <ViennaRNA/structure_utils.h>

Get an exact copy of a pair table.

**Parameters**

| | |
|---|---|
| *pt* | The pair table to be copied |

**Returns**

A pointer to the copy of 'pt'

**15.59.4.17 vrna_pt_snoop_get()**

```
short* vrna_pt_snoop_get (
            const char * structure )
```

#include <ViennaRNA/structure_utils.h>

Create a pair table of a secondary structure (snoop version)

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.

**15.59.4.18 vrna_db_flatten()**

```
vrna_db_flatten (
            char * string,
            unsigned int options )
```

#include <ViennaRNA/structure_utils.h>

Substitute pairs of brackets in a string with parenthesis.

This function can be used to replace brackets of unusual types, such as angular brackets $<>$ , to dot-bracket format. The `options` parameter is used tpo specify which types of brackets will be replaced by round parenthesis () .

**See also**

vrna_db_flatten_to(), VRNA_BRACKETS_RND, VRNA_BRACKETS_ANG, VRNA_BRACKETS_CLY, VRNA_BRACKETS_SQR, VRNA_BRACKETS_DEFAULT

**Parameters**

| structure | The structure string where brackets are flattened in-place |
|---|---|
| options | A bitmask to specify which types of brackets should be flattened out |

**[SWIG Wrapper Notes](#)** This function flattens an input structure string in-place! The second parameter is optional and defaults to VRNA_BRACKETS_DEFAULT.

An overloaded version of this function exists, where an additional second parameter can be passed to specify the target brackets, i.e. the type of matching pair characters all brackets will be flattened to. Therefore, in the scripting language interface this function is a replacement for vrna_db_flatten_to().

**15.59.4.19 vrna_db_flatten_to()**

```
void vrna_db_flatten_to (
            char * string,
            const char target[3],
            unsigned int options )
```

`#include <ViennaRNA/structure_utils.h>`

Substitute pairs of brackets in a string with another type of pair characters.

This function can be used to replace brackets in a structure annotation string, such as square brackets [] , to another type of pair characters, e.g. angular brackets <> .

The `target` array must contain a character for the 'pair open' annotation at position 0, and one for 'pair close' at position 1. T`options` parameter is used to specify which types of brackets will be replaced by the new pairs.

**See also**

> vrna_db_flatten(), VRNA_BRACKETS_RND, VRNA_BRACKETS_ANG, VRNA_BRACKETS_CLY, VRNA_BRACKETS_SQR,
> VRNA_BRACKETS_DEFAULT

**Parameters**

| structure | The structure string where brackets are flattened in-place |
|---|---|
| target | The new pair characters the string will be flattened to |
| options | A bitmask to specify which types of brackets should be flattened out |

**SWIG Wrapper Notes**  This function is available as an overloaded version of vrna_db_flatten()

**15.59.4.20 vrna_db_from_ptable()**

```
char* vrna_db_from_ptable (
            short * pt )
```

`#include <ViennaRNA/structure_utils.h>`

Convert a pair table into dot-parenthesis notation.

**Parameters**

| pt | The pair table to be copied |
|---|---|

**Returns**

> A char pointer to the dot-bracket string

**15.59.4.21 vrna_db_from_WUSS()**

```
char* vrna_db_from_WUSS (
            const char * wuss )
```

#include <ViennaRNA/structure_utils.h>

Convert a WUSS annotation string to dot-bracket format.

**Note**

This function flattens all brackets, and treats pseudo-knots annotated by matching pairs of upper/lowercase letters as unpaired nucleotides

**See also**

Washington University Secondary Structure (WUSS) notation

**Parameters**

| | |
|---|---|
| *wuss* | The input string in WUSS notation |

**Returns**

A dot-bracket notation of the input secondary structure

**15.59.4.22 vrna_bp_distance()**

```
int vrna_bp_distance (
            const char * str1,
            const char * str2 )
```

#include <ViennaRNA/structure_utils.h>

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

**15.59.4.23 vrna_refBPcnt_matrix()**

```
unsigned int* vrna_refBPcnt_matrix (
            const short * reference_pt,
            unsigned int turn )
```

#include <ViennaRNA/structure_utils.h>

Make a reference base pair count matrix.

Get an upper triangular matrix containing the number of basepairs of a reference structure for each interval [i,j] with i<j. Access it via iindx!!!

**15.59.4.24 vrna_refBPdist_matrix()**

```
unsigned int* vrna_refBPdist_matrix (
            const short * pt1,
            const short * pt2,
            unsigned int turn )
```

#include <ViennaRNA/structure_utils.h>

Make a reference base pair distance matrix.

Get an upper triangular matrix containing the base pair distance of two reference structures for each interval [i,j] with i<j. Access it via iindx!!!

**15.59.4.25 vrna_db_from_bp_stack()**

```
char* vrna_db_from_bp_stack (
            vrna_bp_stack_t * bp,
            unsigned int length )
```

#include <ViennaRNA/structure_utils.h>

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

This function is capable to create dot-bracket structures from suboptimal structure prediction sensu M. Zuker

**Parameters**

| | |
|---|---|
| *bp* | Base pair stack containing the traced base pairs |
| *length* | The length of the structure |

**Returns**

The secondary structure in dot-bracket notation as provided in the input

**15.59.4.26 vrna_plist()**

vrna_ep_t* vrna_plist (
       const char * *struc,*
       float *pr* )

#include <ViennaRNA/structure_utils.h>

Create a vrna_ep_t 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

**Parameters**

| | |
|---|---|
| *struc* | The secondary structure in dot-bracket notation |
| *pr* | The probability for each base pair used in the plist |

**Returns**

    The plist array

**15.59.4.27 vrna_db_from_plist()**

char* vrna_db_from_plist (
       vrna_ep_t * *pairs,*
       unsigned int *n* )

#include <ViennaRNA/structure_utils.h>

Convert a list of base pairs into dot-bracket notation.

**See also**

    vrna_plist()

**Parameters**

| | |
|---|---|
| *pairs* | A vrna_ep_t containing the pairs to be included in the dot-bracket string |
| *n* | The length of the structure (number of nucleotides) |

**Returns**

    The dot-bracket string containing the provided base pairs

**15.59.4.28   assign_plist_from_db()**

```
void assign_plist_from_db (
            vrna_ep_t ** pl,
            const char * struc,
            float pr )
```

#include <ViennaRNA/structure_utils.h>

Create a vrna_ep_t 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

**Deprecated**  Use vrna_plist() instead

**Parameters**

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

**15.59.4.29   pack_structure()**

```
char* pack_structure (
            const char * struc )
```

#include <ViennaRNA/structure_utils.h>

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.

**Deprecated**  Use vrna_db_pack() as a replacement

**Parameters**

| struc | The secondary structure in dot-bracket notation |
|---|---|

**Returns**

The binary encoded structure

**15.59.4.30 unpack_structure()**

```
char* unpack_structure (
            const char * packed )
```

`#include <`ViennaRNA/structure_utils.h`>`

Unpack secondary structure previously packed with pack_structure()

Translate a compressed binary string produced by pack_structure() back into the familiar dot-bracket notation.

**Deprecated** Use vrna_db_unpack() as a replacement

**Parameters**

| | |
|---|---|
| *packed* | The binary encoded packed secondary structure |

**Returns**

    The unpacked secondary structure in dot-bracket notation

**15.59.4.31 make_pair_table()**

```
short* make_pair_table (
            const char * structure )
```

`#include <`ViennaRNA/structure_utils.h`>`

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.

**Deprecated** Use vrna_ptable() instead

**Parameters**

| | |
|---|---|
| *structure* | The secondary structure in dot-bracket notation |

**Returns**

    A pointer to the created pair_table

**15.59.4.32   copy_pair_table()**

```
short* copy_pair_table (
            const short * pt )
```

#include <ViennaRNA/structure_utils.h>

Get an exact copy of a pair table.

**Deprecated** Use vrna_ptable_copy() instead

**Parameters**

| pt | The pair table to be copied |
|----|------------------------------|

**Returns**

A pointer to the copy of 'pt'

**15.59.4.33   alimake_pair_table()**

```
short* alimake_pair_table (
            const char * structure )
```

#include <ViennaRNA/structure_utils.h>

Pair table for snoop align

**Deprecated** Use vrna_pt_ali_get() instead!

**15.59.4.34   make_pair_table_snoop()**

```
short* make_pair_table_snoop (
            const char * structure )
```

#include <ViennaRNA/structure_utils.h>

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.

**Deprecated** Use vrna_pt_snoop_get() instead!

**15.59.4.35 bp_distance()**

```
int bp_distance (
            const char * str1,
            const char * str2 )
```

#include <ViennaRNA/structure_utils.h>

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

**Deprecated** Use vrna_bp_distance instead

**Parameters**

| str1 | First structure in dot-bracket notation |
|------|------------------------------------------|
| str2 | Second structure in dot-bracket notation |

**Returns**

The base pair distance between str1 and str2

**15.59.4.36 make_referenceBP_array()**

```
unsigned int* make_referenceBP_array (
            short * reference_pt,
            unsigned int turn )
```

#include <ViennaRNA/structure_utils.h>

Make a reference base pair count matrix.

Get an upper triangular matrix containing the number of basepairs of a reference structure for each interval [i,j] with i<j. Access it via iindx!!!

**Deprecated** Use vrna_refBPcnt_matrix() instead

**15.59.4.37 compute_BPdifferences()**

```
unsigned int* compute_BPdifferences (
            short * pt1,
            short * pt2,
            unsigned int turn )
```

#include <ViennaRNA/structure_utils.h>

Make a reference base pair distance matrix.

Get an upper triangular matrix containing the base pair distance of two reference structures for each interval [i,j] with i<j. Access it via iindx!!!

**Deprecated** Use vrna_refBPdist_matrix() instead

**15.59.4.38 parenthesis_structure()**

```
void parenthesis_structure (
            char * structure,
            vrna_bp_stack_t * bp,
            int length )
```

#include <ViennaRNA/structure_utils.h>

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

**Deprecated** use vrna_parenthesis_structure() instead

**Note**

This function is threadsafe

**15.59.4.39 parenthesis_zuker()**

```
void parenthesis_zuker (
            char * structure,
            vrna_bp_stack_t * bp,
            int length )
```

#include <ViennaRNA/structure_utils.h>

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

**Deprecated** use vrna_parenthesis_zuker instead

**Note**

This function is threadsafe

**15.59.4.40 bppm_to_structure()**

```
void bppm_to_structure (
            char * structure,
            FLT_OR_DBL * pr,
            unsigned int length )
```

`#include <ViennaRNA/structure_utils.h>`

Create a dot-bracket like structure string from base pair probability matrix.

**Deprecated** Use vrna_db_from_probs() instead!

**15.59.4.41 bppm_symbol()**

```
char bppm_symbol (
            const float * x )
```

`#include <ViennaRNA/structure_utils.h>`

Get a pseudo dot bracket notation for a given probability information.

**Deprecated** Use vrna_bpp_symbol() instead!

## 15.60 Utilities for sequence alignments

### 15.60.1 Detailed Description

Collaboration diagram for Utilities for sequence alignments:

**Data Structures**

- struct vrna_pinfo_s

    *A base pair info structure. More...*

**Macros**

- #define VRNA_MEASURE_SHANNON_ENTROPY 1U

    *Flag indicating Shannon Entropy measure.*

**Typedefs**

- typedef struct vrna_pinfo_s vrna_pinfo_t

    *Typename for the base pair info repesenting data structure vrna_pinfo_s.*
- typedef struct vrna_pinfo_s pair_info

    *Old typename of vrna_pinfo_s.*

**Functions**

- int vrna_aln_mpi (const char ∗∗alignment)

    *Get the mean pairwise identity in steps from ?to?(ident)*
- vrna_pinfo_t ∗ vrna_aln_pinfo (vrna_fold_compound_t ∗vc, const char ∗structure, double threshold)

    *Retrieve an array of vrna_pinfo_t structures from precomputed pair probabilities.*
- char ∗∗ vrna_aln_slice (const char ∗∗alignment, unsigned int i, unsigned int j)

    *Slice out a subalignment from a larger alignment.*
- void vrna_aln_free (char ∗∗alignment)

    *Free memory occupied by a set of aligned sequences.*
- char ∗∗ vrna_aln_uppercase (const char ∗∗alignment)

    *Create a copy of an alignment with only uppercase letters in the sequences.*
- char ∗∗ vrna_aln_toRNA (const char ∗∗alignment)

    *Create a copy of an alignment where DNA alphabet is replaced by RNA alphabet.*
- char ∗∗ vrna_aln_copy (const char ∗∗alignment, unsigned int options)

    *Make a copy of a multiple sequence alignment.*
- float ∗ vrna_aln_conservation_struct (const char ∗∗alignment, const char ∗structure, const vrna_md_t ∗md)

    *Compute base pair conservation of a consensus structure.*
- float ∗ vrna_aln_conservation_col (const char ∗∗alignment, const vrna_md_t ∗md_p, unsigned int options)

    *Compute nucleotide conservation in an alignment.*
- int get_mpi (char ∗Alseq[ ], int n_seq, int length, int ∗mini)

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

- 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.*

## 15.60.2 Data Structure Documentation

### 15.60.2.1 struct vrna_pinfo_s

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.

**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*

## 15.60.3 Macro Definition Documentation

**15.60.3.1 VRNA_MEASURE_SHANNON_ENTROPY**

```
#define VRNA_MEASURE_SHANNON_ENTROPY 1U
```

#include <ViennaRNA/aln_util.h>

Flag indicating Shannon Entropy measure.

Shannon Entropy is defined as $H = -\sum_c p_c \cdot \log_2 p_c$

**15.60.4 Typedef Documentation**

**15.60.4.1 pair_info**

```
typedef struct vrna_pinfo_s pair_info
```

#include <ViennaRNA/aln_util.h>

Old typename of vrna_pinfo_s.

**Deprecated** Use vrna_pinfo_t instead!

**15.60.5 Function Documentation**

**15.60.5.1 vrna_aln_mpi()**

```
int vrna_aln_mpi (
            const char ** alignment )
```

#include <ViennaRNA/aln_util.h>

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

**Parameters**

| | |
|---|---|
| *alignment* | Aligned sequences |

**Returns**

The mean pairwise identity

**15.60.5.2 vrna_aln_pinfo()**

[vrna_pinfo_t](#)* vrna_aln_pinfo (
            [vrna_fold_compound_t](#) * *vc,*
            const char * *structure,*
            double *threshold* )

#include <[ViennaRNA/aln_util.h](#)>

Retrieve an array of [vrna_pinfo_t](#) structures from precomputed pair probabilities.

This array of structures contains information about positionwise pair probabilies, base pair entropy and more

**See also**

[vrna_pinfo_t](#), and [vrna_pf()](#)

**Parameters**

| | |
|---|---|
| *vc* | The [vrna_fold_compound_t](#) of type [VRNA_FC_TYPE_COMPARATIVE](#) with precomputed partition function matrices |
| *structure* | An optional structure in dot-bracket notation (Maybe NULL) |
| *threshold* | Do not include results with pair probabilities below threshold |

**Returns**

The [vrna_pinfo_t](#) array

**15.60.5.3 vrna_aln_slice()**

char** vrna_aln_slice (
            const char ** *alignment,*
            unsigned int *i,*
            unsigned int *j* )

#include <[ViennaRNA/aln_util.h](#)>

Slice out a subalignment from a larger alignment.

**Note**

The user is responsible to free the memory occupied by the returned subalignment

**See also**

[vrna_aln_free()](#)

**Parameters**

| | |
|---|---|
| *alignment* | The input alignment |
| *i* | The first column of the subalignment (1-based) |
| *j* | The last column of the subalignment (1-based) |

**Returns**

The subalignment between column $i$ and $j$

**15.60.5.4  vrna_aln_free()**

```
void vrna_aln_free (
              char ** alignment )
```

#include <[ViennaRNA/aln_util.h](ViennaRNA/aln_util.h)>

Free memory occupied by a set of aligned sequences.

**Parameters**

| | |
|---|---|
| *alignment* | The input alignment |

**15.60.5.5  vrna_aln_uppercase()**

```
char** vrna_aln_uppercase (
              const char ** alignment )
```

#include <[ViennaRNA/aln_util.h](ViennaRNA/aln_util.h)>

Create a copy of an alignment with only uppercase letters in the sequences.

**See also**

[vrna_aln_copy](vrna_aln_copy)

**Parameters**

| | |
|---|---|
| *alignment* | The input sequence alignment (last entry must be *NULL* terminated) |

**Returns**

A copy of the input alignment where lowercase sequence letters are replaced by uppercase letters

---

**15.60.5.6 vrna_aln_toRNA()**

```
char** vrna_aln_toRNA (
            const char ** alignment )
```

#include <[ViennaRNA/aln_util.h](ViennaRNA/aln_util.h)>

Create a copy of an alignment where DNA alphabet is replaced by RNA alphabet.

**See also**

>   [vrna_aln_copy](vrna_aln_copy)

**Parameters**

| | |
|---|---|
| *alignment* | The input sequence alignment (last entry must be *NULL* terminated) |

**Returns**

>   A copy of the input alignment where DNA alphabet is replaced by RNA alphabet (T -> U)

**15.60.5.7 vrna_aln_copy()**

```
char** vrna_aln_copy (
            const char ** alignment,
            unsigned int options )
```

#include <[ViennaRNA/aln_util.h](ViennaRNA/aln_util.h)>

Make a copy of a multiple sequence alignment.

This function allows one to create a copy of a multiple sequence alignment. The `options` parameter additionally allows for sequence manipulation, such as converting DNA to RNA alphabet, and conversion to uppercase letters.

**See also**

>   [vrna_aln_copy()](vrna_aln_copy()), #VRNA_ALN_RNA, #VRNA_ALN_UPPERCASE, #VRNA_ALN_DEFAULT

**Parameters**

| | |
|---|---|
| *alignment* | The input sequence alignment (last entry must be *NULL* terminated) |
| *options* | Option flags indicating whether the aligned sequences should be converted |

**Returns**

>   A (manipulated) copy of the input alignment

**15.60.5.8    vrna_aln_conservation_struct()**

```
float * vrna_aln_conservation_struct (
            const char ** alignment,
            const char * structure,
            const vrna_md_t * md )
```

```
#include <ViennaRNA/aln_util.h>
```

Compute base pair conservation of a consensus structure.

This function computes the base pair conservation (fraction of canonical base pairs) of a consensus structure given a multiple sequence alignment. The base pair types that are considered canonical may be specified using the #vrna_md_t.pairs array. Passing *NULL* as parameter md results in default pairing rules, i.e. canonical Watson-Crick and GU Wobble pairs.

**Parameters**

| *alignment* | The input sequence alignment (last entry must be *NULL* terminated) |
|---|---|
| *structure* | The consensus structure in dot-bracket notation |
| *md* | Model details that specify compatible base pairs (Maybe *NULL*) |

**Returns**

A 1-based vector of base pair conservations

**SWIG Wrapper Notes**  This function is available in an overloaded form where the last parameter may be omitted, indicating md = *NULL*

**15.60.5.9    vrna_aln_conservation_col()**

```
float * vrna_aln_conservation_col (
            const char ** alignment,
            const vrna_md_t * md,
            unsigned int options )
```

```
#include <ViennaRNA/aln_util.h>
```

Compute nucleotide conservation in an alignment.

This function computes the conservation of nucleotides in alignment columns. The simples measure is Shannon Entropy and can be selected by passing the VRNA_MEASURE_SHANNON_ENTROPY flag in the options parameter.

**Note**

Currently, only VRNA_MEASURE_SHANNON_ENTROPY is supported as conservation measure.

**See also**

VRNA_MEASURE_SHANNON_ENTROPY

**Parameters**

| *alignment* | The input sequence alignment (last entry must be *NULL* terminated) |
|---|---|
| *md* | Model details that specify known nucleotides (Maybe *NULL*) |
| *options* | A flag indicating which measure of conservation should be applied |

**Returns**

A 1-based vector of column conservations

**SWIG Wrapper Notes** This function is available in an overloaded form where the last two parameters may be omitted, indicating md = *NULL*, and `options` = VRNA_MEASURE_SHANNON_ENTROPY, respectively.

**15.60.5.10 get_mpi()**

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

`#include <`ViennaRNA/aln_util.h`>`

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

**Deprecated** Use vrna_aln_mpi() as a replacement

**Parameters**

| *Alseq* | |
|---|---|
| *n_seq* | The number of sequences in the alignment |
| *length* | The length of the alignment |
| *mini* | |

**Returns**

The mean pairwise identity

**15.60.5.11 encode_ali_sequence()**

```
void encode_ali_sequence (
            const char * sequence,
```

```
            short * S,
            short * s5,
            short * s3,
            char * ss,
            unsigned short * as,
            int circ )
```

#include <ViennaRNA/aln_util.h>

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) |

**15.60.5.12 alloc_sequence_arrays()**

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

#include <ViennaRNA/aln_util.h>

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) |

**15.60.5.13   free_sequence_arrays()**

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

`#include <`ViennaRNA/aln_util.h`>`

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 |

## 15.61 Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments

Functions dealing with file formats for RNA sequences, structures, and alignments.

### 15.61.1 Detailed Description

Functions dealing with file formats for RNA sequences, structures, and alignments.

Collaboration diagram for Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments:

### Files

- file commands.h

  *Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.*

- file file_formats.h

  *Read and write different file formats for RNA sequences, structures.*

- file file_formats_msa.h

  *Functions dealing with file formats for Multiple Sequence Alignments (MSA)*

- file file_utils.h

  *Several utilities for file handling.*

- file ribo.h

  *Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.*

### Data Structures

- struct vrna_command_s

  *List element for commands ready for application to a vrna_fold_compound_t. More...*

### Macros

- #define VRNA_CMD_PARSE_HC 1U

  *Command parse/apply flag indicating hard constraints.*

- #define VRNA_CMD_PARSE_SC 2U

  *Command parse/apply flag indicating soft constraints.*

- #define VRNA_CMD_PARSE_UD 4U

  *Command parse/apply flag indicating unstructured domains.*

- #define VRNA_CMD_PARSE_SD 8U

  *Command parse/apply flag indicating structured domains.*

- #define VRNA_CMD_PARSE_DEFAULTS

  *Command parse/apply flag indicating default set of commands.*

- #define VRNA_OPTION_MULTILINE 32U

  *Tell a function that an input is assumed to span several lines.*

- #define VRNA_CONSTRAINT_MULTILINE 32U

*parse multiline constraint*

- #define VRNA_FILE_FORMAT_MSA_CLUSTAL 1U

  *Option flag indicating ClustalW formatted files.*

- #define VRNA_FILE_FORMAT_MSA_STOCKHOLM 2U

  *Option flag indicating Stockholm 1.0 formatted files.*

- #define VRNA_FILE_FORMAT_MSA_FASTA 4U

  *Option flag indicating FASTA (Pearson) formatted files.*

- #define VRNA_FILE_FORMAT_MSA_MAF 8U

  *Option flag indicating MAF formatted files.*

- #define VRNA_FILE_FORMAT_MSA_MIS 16U

  *Option flag indicating most informative sequence (MIS) output.*

- #define VRNA_FILE_FORMAT_MSA_DEFAULT

  *Option flag indicating the set of default file formats.*

- #define VRNA_FILE_FORMAT_MSA_NOCHECK 4096U

  *Option flag to disable validation of the alignment.*

- #define VRNA_FILE_FORMAT_MSA_UNKNOWN 8192U

  *Return flag of vrna_file_msa_detect_format() to indicate unknown or malformatted alignment.*

- #define VRNA_FILE_FORMAT_MSA_APPEND 16384U

  *Option flag indicating to append data to a multiple sequence alignment file rather than overwriting it.*

- #define VRNA_FILE_FORMAT_MSA_QUIET 32768U

  *Option flag to suppress unnecessary spam messages on* `stderr`

- #define VRNA_FILE_FORMAT_MSA_SILENT 65536U

  *Option flag to completely silence any warnings on* `stderr`

## Typedefs

- typedef struct vrna_command_s vrna_cmd_t

  *Typename for the command repesenting data structure vrna_command_s.*

## Enumerations

- enum vrna_command_e

  *Types of commands within a list of vrna_command_s structures.*

## Functions

- vrna_cmd_t ∗ vrna_file_commands_read (const char ∗filename, unsigned int options)

  *Extract a list of commands from a command file.*

- int vrna_file_commands_apply (vrna_fold_compound_t ∗vc, const char ∗filename, unsigned int options)

  *Apply a list of commands from a command file.*

- int vrna_commands_apply (vrna_fold_compound_t ∗vc, vrna_cmd_t ∗commands, unsigned int options)

  *Apply a list of commands to a vrna_fold_compound_t.*

- void vrna_commands_free (vrna_cmd_t ∗commands)

  *Free memory occupied by a list of commands.*

- void vrna_file_helixlist (const char ∗seq, const char ∗db, float energy, FILE ∗file)

  *Print a secondary structure as helix list.*

- void vrna_file_connect (const char ∗seq, const char ∗db, float energy, const char ∗identifier, FILE ∗file)

  *Print a secondary structure as connect table.*

- void vrna_file_bpseq (const char ∗seq, const char ∗db, FILE ∗file)

*Print a secondary structure in bpseq format.*

• void vrna_file_json (const char ∗seq, const char ∗db, double energy, const char ∗identifier, FILE ∗file)

*Print a secondary structure in jsonformat.*

• unsigned int vrna_file_fasta_read_record (char ∗∗header, char ∗∗sequence, char ∗∗∗rest, FILE ∗file, unsigned int options)

*Get a (fasta) data set from a file or stdin.*

• char ∗ vrna_extract_record_rest_structure (const char ∗∗lines, unsigned int length, unsigned int option)

*Extract a dot-bracket structure string from (multiline)character array.*

• int vrna_file_SHAPE_read (const char ∗file_name, int length, double default_value, char ∗sequence, double ∗values)

*Read data from a given SHAPE reactivity input file.*

• void vrna_extract_record_rest_constraint (char ∗∗cstruc, const char ∗∗lines, unsigned int option)

*Extract a hard constraint encoded as pseudo dot-bracket string.*

• unsigned int read_record (char ∗∗header, char ∗∗sequence, char ∗∗∗rest, unsigned int options)

*Get a data record from stdin.*

• int vrna_file_msa_read (const char ∗filename, char ∗∗∗names, char ∗∗∗aln, char ∗∗id, char ∗∗structure, unsigned int options)

*Read a multiple sequence alignment from file.*

• int vrna_file_msa_read_record (FILE ∗fp, char ∗∗∗names, char ∗∗∗aln, char ∗∗id, char ∗∗structure, unsigned int options)

*Read a multiple sequence alignment from file handle.*

• unsigned int vrna_file_msa_detect_format (const char ∗filename, unsigned int options)

*Detect the format of a multiple sequence alignment file.*

• int vrna_file_msa_write (const char ∗filename, const char ∗∗names, const char ∗∗aln, const char ∗id, const char ∗structure, const char ∗source, unsigned int options)

*Write multiple sequence alignment file.*

• void vrna_file_copy (FILE ∗from, FILE ∗to)

*Inefficient 'cp'.*

• char ∗ vrna_read_line (FILE ∗fp)

*Read a line of arbitrary length from a stream.*

• int vrna_mkdir_p (const char ∗path)

*Recursivly create a directory tree.*

• char ∗ vrna_basename (const char ∗path)

*Extract the filename from a file path.*

• char ∗ vrna_dirname (const char ∗path)

*Extract the directory part of a file path.*

• char ∗ vrna_filename_sanitize (const char ∗name, const char ∗replacement)

*Sanitize a file name.*

• float ∗∗ readribosum (char ∗name)

*Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.*

## 15.61.2   Data Structure Documentation

### 15.61.2.1   struct vrna_command_s

List element for commands ready for application to a vrna_fold_compound_t.

**See also**

vrna_file_commands_read(), vrna_commands_apply(), vrna_commands_free()

### 15.61.3 Macro Definition Documentation

#### 15.61.3.1 VRNA_CMD_PARSE_HC

#define VRNA_CMD_PARSE_HC 1U

#include <ViennaRNA/commands.h>

Command parse/apply flag indicating hard constraints.

**See also**

vrna_command_s, vrna_file_commands_read(), vrna_file_commands_apply(), vrna_commands_apply()

#### 15.61.3.2 VRNA_CMD_PARSE_SC

#define VRNA_CMD_PARSE_SC 2U

#include <ViennaRNA/commands.h>

Command parse/apply flag indicating soft constraints.

**See also**

vrna_command_s, vrna_file_commands_read(), vrna_file_commands_apply(), vrna_commands_apply()

#### 15.61.3.3 VRNA_CMD_PARSE_UD

#define VRNA_CMD_PARSE_UD 4U

#include <ViennaRNA/commands.h>

Command parse/apply flag indicating unstructured domains.

**See also**

vrna_command_s, vrna_file_commands_read(), vrna_file_commands_apply(), vrna_commands_apply()

**15.61.3.4    VRNA_CMD_PARSE_SD**

```
#define VRNA_CMD_PARSE_SD 8U
```

`#include <`ViennaRNA/commands.h`>`

Command parse/apply flag indicating structured domains.

**See also**

vrna_command_s, vrna_file_commands_read(), vrna_file_commands_apply(), vrna_commands_apply()

**15.61.3.5    VRNA_CMD_PARSE_DEFAULTS**

```
#define VRNA_CMD_PARSE_DEFAULTS
```

`#include <`ViennaRNA/commands.h`>`

**Value:**

```
(VRNA_CMD_PARSE_HC \
                            | VRNA_CMD_PARSE_SC \
                            | VRNA_CMD_PARSE_UD \
                            | VRNA_CMD_PARSE_SD \
                            )
```

Command parse/apply flag indicating default set of commands.

**See also**

vrna_command_s, vrna_file_commands_read(), vrna_file_commands_apply(), vrna_commands_apply()

**15.61.3.6    VRNA_OPTION_MULTILINE**

```
#define VRNA_OPTION_MULTILINE 32U
```

`#include <`ViennaRNA/file_formats.h`>`

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**

vrna_extract_record_rest_structure(), vrna_file_fasta_read_record()

**15.61.3.7 VRNA_CONSTRAINT_MULTILINE**

```
#define VRNA_CONSTRAINT_MULTILINE 32U
```

`#include <`ViennaRNA/file_formats.h`>`

parse multiline constraint

**Deprecated** see vrna_extract_record_rest_structure()

**15.61.3.8 VRNA_FILE_FORMAT_MSA_CLUSTAL**

```
#define VRNA_FILE_FORMAT_MSA_CLUSTAL 1U
```

`#include <`ViennaRNA/file_formats_msa.h`>`

Option flag indicating ClustalW formatted files.

**See also**

vrna_file_msa_read(), vrna_file_msa_read_record(), vrna_file_msa_detect_format()

**15.61.3.9 VRNA_FILE_FORMAT_MSA_STOCKHOLM**

```
#define VRNA_FILE_FORMAT_MSA_STOCKHOLM 2U
```

`#include <`ViennaRNA/file_formats_msa.h`>`

Option flag indicating Stockholm 1.0 formatted files.

**See also**

vrna_file_msa_read(), vrna_file_msa_read_record(), vrna_file_msa_detect_format()

**15.61.3.10 VRNA_FILE_FORMAT_MSA_FASTA**

```
#define VRNA_FILE_FORMAT_MSA_FASTA 4U
```

`#include <`ViennaRNA/file_formats_msa.h`>`

Option flag indicating FASTA (Pearson) formatted files.

**See also**

vrna_file_msa_read(), vrna_file_msa_read_record(), vrna_file_msa_detect_format()

**15.61.3.11  VRNA_FILE_FORMAT_MSA_MAF**

```
#define VRNA_FILE_FORMAT_MSA_MAF 8U
```

#include <ViennaRNA/file_formats_msa.h>

Option flag indicating MAF formatted files.

**See also**

vrna_file_msa_read(), vrna_file_msa_read_record(), vrna_file_msa_detect_format()

**15.61.3.12  VRNA_FILE_FORMAT_MSA_MIS**

```
#define VRNA_FILE_FORMAT_MSA_MIS 16U
```

#include <ViennaRNA/file_formats_msa.h>

Option flag indicating most informative sequence (MIS) output.

The default reference sequence output for an alignment is simply a consensus sequence. This flag allows to write the most informative equence (MIS) instead.

**See also**

vrna_file_msa_write()

**15.61.3.13  VRNA_FILE_FORMAT_MSA_DEFAULT**

```
#define VRNA_FILE_FORMAT_MSA_DEFAULT
```

#include <ViennaRNA/file_formats_msa.h>

**Value:**

```
( \
    VRNA_FILE_FORMAT_MSA_CLUSTAL \
    | VRNA_FILE_FORMAT_MSA_STOCKHOLM \
    | VRNA_FILE_FORMAT_MSA_FASTA \
    | VRNA_FILE_FORMAT_MSA_MAF \
    )
```

Option flag indicating the set of default file formats.

**See also**

vrna_file_msa_read(), vrna_file_msa_read_record(), vrna_file_msa_detect_format()

**15.61.3.14 VRNA_FILE_FORMAT_MSA_NOCHECK**

`#define VRNA_FILE_FORMAT_MSA_NOCHECK 4096U`

`#include <`ViennaRNA/file_formats_msa.h`>`

Option flag to disable validation of the alignment.

**See also**

> vrna_file_msa_read(), vrna_file_msa_read_record()

**15.61.3.15 VRNA_FILE_FORMAT_MSA_UNKNOWN**

`#define VRNA_FILE_FORMAT_MSA_UNKNOWN 8192U`

`#include <`ViennaRNA/file_formats_msa.h`>`

Return flag of vrna_file_msa_detect_format() to indicate unknown or malformatted alignment.

**See also**

> vrna_file_msa_detect_format()

**15.61.3.16 VRNA_FILE_FORMAT_MSA_APPEND**

`#define VRNA_FILE_FORMAT_MSA_APPEND 16384U`

`#include <`ViennaRNA/file_formats_msa.h`>`

Option flag indicating to append data to a multiple sequence alignment file rather than overwriting it.

**See also**

> vrna_file_msa_write()

**15.61.3.17 VRNA_FILE_FORMAT_MSA_QUIET**

`#define VRNA_FILE_FORMAT_MSA_QUIET 32768U`

`#include <`ViennaRNA/file_formats_msa.h`>`

Option flag to suppress unnecessary spam messages on `stderr`

**See also**

> vrna_file_msa_read(), vrna_file_msa_read_record()

**15.61.3.18   VRNA_FILE_FORMAT_MSA_SILENT**

```
#define VRNA_FILE_FORMAT_MSA_SILENT 65536U
```

#include <ViennaRNA/file_formats_msa.h>

Option flag to completely silence any warnings on `stderr`

**See also**

vrna_file_msa_read(), vrna_file_msa_read_record()

**15.61.4   Function Documentation**

**15.61.4.1   vrna_file_commands_read()**

```
vrna_cmd_t* vrna_file_commands_read (
            const char * filename,
            unsigned int options )
```

#include <ViennaRNA/commands.h>

Extract a list of commands from a command file.

Read a list of commands specified in the input file and return them as list of abstract commands

**See also**

vrna_commands_apply(), vrna_file_commands_apply(), vrna_commands_free()

**Parameters**

| filename | The filename |
|----------|--------------|
| options | Options to limit the type of commands read from the file |

**Returns**

A list of abstract commands

**15.61.4.2   vrna_file_commands_apply()**

```
int vrna_file_commands_apply (
            vrna_fold_compound_t * vc,
```

```
            const char * filename,
            unsigned int options )
```

`#include <`ViennaRNA/commands.h`>`

Apply a list of commands from a command file.

This function is a shortcut to directly parse a commands file and apply all successfully parsed commands to a vrna_fold_compound_t data structure. It is the same as:

```
int         r;
vrna_cmd_t  *cmds;

cmds  = vrna_file_commands_read(filename, options);
r     = vrna_commands_apply(vc, cmds, options);

vrna_commands_free(cmds);

return r;
```

**Parameters**

| vc | The vrna_fold_compound_t the command list will be applied to |
|---|---|
| filename | The filename |
| options | Options to limit the type of commands read from the file |

**Returns**

The number of commands successfully applied

**SWIG Wrapper Notes** This function is attached as method **file_commands_apply()** to objects of type *fold_↩ compound*

**15.61.4.3 vrna_commands_apply()**

```
int vrna_commands_apply (
            vrna_fold_compound_t * vc,
            vrna_cmd_t * commands,
            unsigned int options )
```

`#include <`ViennaRNA/commands.h`>`

Apply a list of commands to a vrna_fold_compound_t.

**Parameters**

| vc | The vrna_fold_compound_t the command list will be applied to |
|---|---|
| commands | The list of commands to apply |
| options | Options to limit the type of commands read from the file |

**Returns**

The number of commands successfully applied

**SWIG Wrapper Notes** This function is attached as method **commands_apply()** to objects of type *fold_compound*

**15.61.4.4   vrna_commands_free()**

```
void vrna_commands_free (
            vrna_cmd_t * commands )
```

`#include <`ViennaRNA/commands.h`>`

Free memory occupied by a list of commands.

Release memory occupied by a list of commands

**Parameters**

| | |
|---|---|
| *commands* | A pointer to a list of commands |

**15.61.4.5   vrna_file_helixlist()**

```
void vrna_file_helixlist (
            const char * seq,
            const char * db,
            float energy,
            FILE * file )
```

`#include <`ViennaRNA/file_formats.h`>`

Print a secondary structure as helix list.

**Parameters**

| | |
|---|---|
| *seq* | The RNA sequence |
| *db* | The structure in dot-bracket format |
| *energy* | Free energy of the structure in kcal/mol |
| *file* | The file handle used to print to (print defaults to 'stdout' if(file == NULL) ) |

**15.61.4.6   vrna_file_connect()**

```
void vrna_file_connect (
            const char * seq,
```

```
        const char * db,
        float energy,
        const char * identifier,
        FILE * file )
```

#include <ViennaRNA/file_formats.h>

Print a secondary structure as connect table.

Connect table file format looks like this:

```
300  ENERGY = 7.0  example
  1 G      0   2   22    1
  2 G      1   3   21    2
```

where the headerline is followed by 6 columns with:

1. Base number: index n

2. Base (A, C, G, T, U, X)

3. Index n-1 (0 if first nucleotide)

4. Index n+1 (0 if last nucleotide)

5. Number of the base to which n is paired. No pairing is indicated by 0 (zero).

6. Natural numbering.

**Parameters**

| | |
|---|---|
| *seq* | The RNA sequence |
| *db* | The structure in dot-bracket format |
| *energy* | The free energy of the structure |
| *identifier* | An optional identifier for the sequence |
| *file* | The file handle used to print to (print defaults to 'stdout' if(file == NULL) ) |

**15.61.4.7 vrna_file_bpseq()**

```
void vrna_file_bpseq (
        const char * seq,
        const char * db,
        FILE * file )
```

#include <ViennaRNA/file_formats.h>

Print a secondary structure in bpseq format.

**Parameters**

| | |
|---|---|
| *seq* | The RNA sequence |
| *db* | The structure in dot-bracket format |
| *file* | The file handle used to print to (print defaults to 'stdout' if(file == NULL) ) |

**15.61.4.8    vrna_file_json()**

```
void vrna_file_json (
            const char * seq,
            const char * db,
            double energy,
            const char * identifier,
            FILE * file )
```

`#include <`ViennaRNA/file_formats.h`>`

Print a secondary structure in jsonformat.

**Parameters**

| seq | The RNA sequence |
| --- | --- |
| db | The structure in dot-bracket format |
| energy | The free energy |
| identifier | An identifier for the sequence |
| file | The file handle used to print to (print defaults to 'stdout' if(file == NULL) ) |

**15.61.4.9    vrna_file_fasta_read_record()**

```
unsigned int vrna_file_fasta_read_record (
            char ** header,
            char ** sequence,
            char *** rest,
            FILE * file,
            unsigned int options )
```

`#include <`ViennaRNA/file_formats.h`>`

Get a (fasta) data set from a file or stdin.

This function may be used to obtain complete datasets from a filehandle or stdin. A dataset is always defined to contain at least a sequence. If data starts with a fasta header, i.e. a line like

```
>some header info
```

then vrna_file_fasta_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) that do not initiate a new dataset according to the above definition 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!
> This function is NOT threadsafe! It uses a global variable to store information about the next data block.

The main purpose of this function is to be able to easily parse blocks of data 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;
id = seq = NULL;
rest = NULL;
while(!(vrna_file_fasta_read_record(&id, &seq, &rest, NULL, 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]);
      free(rest[i]);
    }
  free(rest);
  free(seq);
  free(id);
}
```

In the example above, the while loop will be terminated when vrna_file_fasta_read_record() returns either an error, EOF, or a user initiated quit request.

As long as data is read from stdin (we are passing NULL as the file pointer), 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 |
| file     | A file handle to read from (if NULL, this function reads from stdin)                           |
| 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

**15.61.4.10   vrna_extract_record_rest_structure()**

```
char* vrna_extract_record_rest_structure (
            const char ** lines,
            unsigned int length,
            unsigned int option )
```

```
#include <ViennaRNA/file_formats.h>
```

Extract a dot-bracket structure string from (multiline)character array.

This function extracts a dot-bracket structure string from the 'rest' array as returned by vrna_file_fasta_read_record() and returns it. All occurences of comments within the 'lines' array will be skipped as long as they do not break the structure string. If no structure could be read, this function returns NULL.

**Precondition**

The argument 'lines' has to be a 2-dimensional character array as obtained by vrna_file_fasta_read_record()

**See also**

vrna_file_fasta_read_record()

**Parameters**

| lines | The (multiline) character array to be parsed |
|---|---|
| length | The assumed length of the dot-bracket string (passing a value $< 1$ results in no length limit) |
| option | Some options which may be passed to alter the behavior of the function, use 0 for no options |

**Returns**

The dot-bracket string read from lines or NULL

**15.61.4.11    vrna_file_SHAPE_read()**

```
int vrna_file_SHAPE_read (
            const char * file_name,
            int length,
            double default_value,
            char * sequence,
            double * values )
```

```
#include <ViennaRNA/file_formats.h>
```

Read data from a given SHAPE reactivity input file.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the double array values.

**Parameters**

| file_name | Path to the constraints file |
|---|---|
| length | Length of the sequence (file entries exceeding this limit will cause an error) |
| default_value | Value for missing indices |
| sequence | Pointer to an array used for storing the sequence obtained from the SHAPE reactivity file |
| values | Pointer to an array used for storing the values obtained from the SHAPE reactivity file |

**15.61.4.12 vrna_extract_record_rest_constraint()**

```
void vrna_extract_record_rest_constraint (
            char ** cstruc,
            const char ** lines,
            unsigned int option )
```

#include <ViennaRNA/file_formats.h>

Extract a hard constraint encoded as pseudo dot-bracket string.

**Deprecated** Use vrna_extract_record_rest_structure() instead!

**Precondition**

The argument 'lines' has to be a 2-dimensional character array as obtained by vrna_file_fasta_read_record()

**See also**

vrna_file_fasta_read_record(), VRNA_CONSTRAINT_DB_PIPE, VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_
VRNA_CONSTRAINT_DB_ANG_BRACK, VRNA_CONSTRAINT_DB_RND_BRACK

**Parameters**

| cstruc | A pointer to a character array that is used as pseudo dot-bracket output |
|--------|--------------------------------------------------------------------------|
| lines  | A 2-dimensional character array with the extension lines from the FASTA input |
| option | The option flags that define the behavior and recognition pattern of this function |

**15.61.4.13 read_record()**

```
unsigned int read_record (
            char ** header,
            char ** sequence,
            char *** rest,
            unsigned int options )
```

#include <ViennaRNA/file_formats.h>

Get a data record from stdin.

**Deprecated** This function is deprecated! Use vrna_file_fasta_read_record() as a replacment.

**15.61.4.14    vrna_file_msa_read()**

```
vrna_file_msa_read (
            const char * filename,
            char *** names,
            char *** aln,
            char ** id,
            char ** structure,
            unsigned int options )
```

#include <ViennaRNA/file_formats_msa.h>

Read a multiple sequence alignment from file.

This function reads the (first) multiple sequence alignment from an input file. The read alignment is split into the sequence id/name part and the actual sequence information and stored in memory as arrays of ids/names and sequences. If the alignment file format allows for additional information, such as an ID of the entire alignment or consensus structure information, this data is retrieved as well and made available. The `options` parameter allows to specify the set of alignment file formats that should be used to retrieve the data. If 0 is passed as option, the list of alignment file formats defaults to VRNA_FILE_FORMAT_MSA_DEFAULT.

Currently, the list of parsable multiple sequence alignment file formats consists of:

- ClustalW format

- Stockholm 1.0 format

- FASTA (Pearson) format

- MAF format

**Note**

> After successfully reading an alignment, this function performs a validation of the data that includes unique-ness of the sequence identifiers, and equal sequence lengths. This check can be deactivated by passing VRNA_FILE_FORMAT_MSA_NOCHECK in the `options` parameter.
> It is the users responsibility to free any memory occupied by the output arguments `names`, `aln`, `id`, and `structure` after calling this function. The function automatically sets the latter two arguments to `NULL` in case no corresponding data could be retrieved from the input alignment.

**See also**

> vrna_file_msa_read_record(), VRNA_FILE_FORMAT_MSA_CLUSTAL, VRNA_FILE_FORMAT_MSA_STOCKHOLM, VRNA_FILE_FORMAT_MSA_FASTA, VRNA_FILE_FORMAT_MSA_MAF, VRNA_FILE_FORMAT_MSA_DEFAULT, VRNA_FILE_FORMAT_MSA_NOCHECK

**Parameters**

| | |
|---|---|
| *filename* | The name of input file that contains the alignment |
| *names* | An address to the pointer where sequence identifiers should be written to |
| *aln* | An address to the pointer where aligned sequences should be written to |
| *id* | An address to the pointer where the alignment ID should be written to (Maybe NULL) |
| *structure* | An address to the pointer where consensus structure information should be written to (Maybe NULL) |
| *options* | Options to manipulate the behavior of this function |

**Returns**

> The number of sequences in the alignment, or -1 if no alignment record could be found

**SWIG Wrapper Notes** In the target scripting language, only the first and last argument, `filename` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
num_seq, names, aln, id, structure = RNA.file_msa_read("msa.stk", RNA.FILE_FORMAT_MSA_STOCKHOLM)
```

> After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

> This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to VRNA_FILE_FORMAT_MSA_STOCKHOLM.

**15.61.4.15 vrna_file_msa_read_record()**

```
vrna_file_msa_read_record (
            FILE * fp,
            char *** names,
            char *** aln,
            char ** id,
            char ** structure,
            unsigned int options )
```

`#include <ViennaRNA/file_formats_msa.h>`

Read a multiple sequence alignment from file handle.

Similar to vrna_file_msa_read(), this function reads a multiple sequence alignment from an input file handle. Since using a file handle, this function is not limited to the first alignment record, but allows for looping over all alignments within the input.

The read alignment is split into the sequence id/name part and the actual sequence information and stored in memory as arrays of ids/names and sequences. If the alignment file format allows for additional information, such as an ID of the entire alignment or consensus structure information, this data is retrieved as well and made available. The `options` parameter allows to specify the alignment file format used to retrieve the data. A single format must be specified here, see vrna_file_msa_detect_format() for helping to determine the correct MSA file format.

Currently, the list of parsable multiple sequence alignment file formats consists of:

- ClustalW format

- Stockholm 1.0 format

- FASTA (Pearson) format

- MAF format

**Note**

> After successfully reading an alignment, this function performs a validation of the data that includes unique-
> ness of the sequence identifiers, and equal sequence lengths. This check can be deactivated by passing
> VRNA_FILE_FORMAT_MSA_NOCHECK in the `options` parameter.
> It is the users responsibility to free any memory occupied by the output arguments `names`, `aln`, `id`, and
> `structure` after calling this function. The function automatically sets the latter two arguments to `NULL` in
> case no corresponding data could be retrieved from the input alignment.

**See also**

> vrna_file_msa_read(), vrna_file_msa_detect_format(), VRNA_FILE_FORMAT_MSA_CLUSTAL, VRNA_FILE_FORMAT_MSA_
> VRNA_FILE_FORMAT_MSA_FASTA, VRNA_FILE_FORMAT_MSA_MAF, VRNA_FILE_FORMAT_MSA_DEFAULT,
> VRNA_FILE_FORMAT_MSA_NOCHECK

**Parameters**

| | |
|---|---|
| *fp* | The file pointer the data will be retrieved from |
| *names* | An address to the pointer where sequence identifiers should be written to |
| *aln* | An address to the pointer where aligned sequences should be written to |
| *id* | An address to the pointer where the alignment ID should be written to (Maybe NULL) |
| *structure* | An address to the pointer where consensus structure information should be written to (Maybe NULL) |
| *options* | Options to manipulate the behavior of this function |

**Returns**

> The number of sequences in the alignment, or -1 if no alignment record could be found

**SWIG Wrapper Notes**  In the target scripting language, only the first and last argument, `fp` and `options`, are
passed to the corresponding function. The other arguments, which serve as output in the
C-library, are available as additional return values. Hence, a function call in python may
look like this:

```
f = open('msa.stk', 'r')
num_seq, names, aln, id, structure = RNA.file_msa_read_record(f, RNA.FILE_FORMAT_MSA_STOCKHOLM)
f.close()
```

After successfully reading the first record, the variable `num_seq` contains the number of
sequences in the alignment (the actual return value of the C-function), while the variables
`names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned se-
quences, as well as strings holding the alignment ID and the structure as stated in the
`SS_cons` line, respectively. Note, the last two return values may be empty strings in case
the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omit-
ted! In that case, the `options` parameter defaults to VRNA_FILE_FORMAT_MSA_STOCKHOLM.

**15.61.4.16   vrna_file_msa_detect_format()**

```
vrna_file_msa_detect_format (
          const char * fn,
          unsigned int options )
```

```
#include <ViennaRNA/file_formats_msa.h>
```

Detect the format of a multiple sequence alignment file.

This function attempts to determine the format of a file that supposedly contains a multiple sequence alignment (M←SA). This is useful in cases where a MSA file contains more than a single record and therefore vrna_file_msa_read() can not be applied, since it only retrieves the first. Here, one can try to guess the correct file format using this function and then loop over the file, record by record using one of the low-level record retrieval functions for the corresponding MSA file format.

**Note**

> This function parses the entire first record within the specified file. As a result, it returns VRNA_FILE_FORMAT_MSA_UNKNOWN not only if it can't detect the file's format, but also in cases where the file doesn't contain sequences!

**See also**

> vrna_file_msa_read(), vrna_file_stockholm_read_record(), vrna_file_clustal_read_record(), vrna_file_fasta_read_record()

**Parameters**

| | |
|---|---|
| *filename* | The name of input file that contains the alignment |
| *options* | Options to manipulate the behavior of this function |

**Returns**

> The MSA file format, or VRNA_FILE_FORMAT_MSA_UNKNOWN

**SWIG Wrapper Notes** This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to VRNA_FILE_FORMAT_MSA_DEFAULT.

**15.61.4.17 vrna_file_msa_write()**

```
vrna_file_msa_write (
            const char * filename,
            const char ** names,
            const char ** aln,
            const char * id,
            const char * structure,
            const char * source,
            unsigned int options )
```

```
#include <ViennaRNA/file_formats_msa.h>
```

Write multiple sequence alignment file.

**Note**

> Currently, we only support Stockholm 1.0 format output

**See also**

> VRNA_FILE_FORMAT_MSA_STOCKHOLM, VRNA_FILE_FORMAT_MSA_APPEND, VRNA_FILE_FORMAT_MSA_MIS

**Parameters**

| filename | The output filename |
|---|---|
| names | The array of sequence names / identifies |
| aln | The array of aligned sequences |
| id | An optional ID for the alignment |
| structure | An optional consensus structure |
| source | A string describing the source of the alignment |
| options | Options to manipulate the behavior of this function |

**Returns**

Non-null upon successfully writing the alignment to file

**SWIG Wrapper Notes** In the target scripting language, this function exists as a set of overloaded versions, where the last four parameters may be omitted. If the `options` parameter is missing the options default to (VRNA_FILE_FORMAT_MSA_STOCKHOLM | VRNA_FILE_FORMAT_MSA_APPEND).

**15.61.4.18   vrna_read_line()**

```
char* vrna_read_line (
            FILE * fp )
```

```
#include <ViennaRNA/file_utils.h>
```

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

**15.61.4.19   vrna_filename_sanitize()**

```
char* vrna_filename_sanitize (
            const char * name,
            const char * replacement )
```

#include <ViennaRNA/file_utils.h>

Sanitize a file name.

Returns a new file name where all invalid characters are substituted by a replacement character. If no replacement character is supplied, invalid characters are simply removed from the filename. File names may also never exceed a length of 255 characters. Longer file names will undergo a 'smart' truncation process, where the filenames' suffix, i.e. everything after the last dot '.', is attempted to be kept intact. Hence, only the filename part before the suffix is reduced in such a way that the total filename complies to the length restriction of 255 characters. If no suffix is present or the suffix itself already exceeds the maximum length, the filename is simply truncated from the back of the string.

For now we consider the following characters invalid:

- backslash '\'

- slash '/'

- question mark '?'

- percent sign "

- asterisk '∗'

- colon ':'

- pipe symbol '|'

- double quote '"'

- triangular brackets '<' and '>'

Furthermore, the (resulting) file name must not be a reserved file name, such as:

- '.'

- '..'

**Note**

> This function allocates a new block of memory for the sanitized string. It also may return (a) NULL if the input is pointing to NULL, or (b) an empty string if the input only consists of invalid characters which are simply removed!

**Parameters**

| name | The input file name |
|------|---------------------|
| replacment | The replacement character, or NULL |

**Returns**

> The sanitized file name, or NULL

#include <ViennaRNA/file_utils.h>

## 15.62 Functions that provide dynamically buffered stream-like data structures

### 15.62.1 Detailed Description

Collaboration diagram for Functions that provide dynamically buffered stream-like data structures:

### Files

- file char_stream.h

  *Implementation of a dynamic, buffered character stream.*
- file stream_output.h

  *An implementation of a buffered, ordered stream output data structure.*

### Typedefs

- typedef struct vrna_ordered_stream_s ∗ vrna_ostream_t

  *An ordered output stream structure with unordered insert capabilities.*
- typedef void() vrna_callback_stream_output(void ∗auxdata, unsigned int i, void ∗data)

  *Ordered stream processing callback.*

### Functions

- vrna_ostream_t vrna_ostream_init (vrna_callback_stream_output ∗output, void ∗auxdata)

  *Get an initialized ordered output stream.*
- void vrna_ostream_free (vrna_ostream_t dat)

  *Free an initialized ordered output stream.*
- void vrna_ostream_request (vrna_ostream_t dat, unsigned int num)

  *Request index in ordered output stream.*
- void vrna_ostream_provide (vrna_ostream_t dat, unsigned int i, void ∗data)

  *Provide output stream data for a particular index.*

### 15.62.2 Typedef Documentation

#### 15.62.2.1 vrna_callback_stream_output

```
typedef void() vrna_callback_stream_output(void *auxdata, unsigned int i, void *data)
```

`#include <ViennaRNA/stream_output.h>`

Ordered stream processing callback.

This callback will be processed in sequential order as soon as sequential data in the output stream becomes available.

**Note**

The callback must also release the memory occupied by the data passed since the stream will lose any reference to it after the callback has been executed.

**Parameters**

| auxdata | A shared pointer for all calls, as provided by the second argument to vrna_ostream_init() |
|---------|-------------------------------------------------------------------------------------------|
| i | The index number of the data passed to `data` |
| data | A block of data ready for processing |

## 15.62.3 Function Documentation

### 15.62.3.1 vrna_ostream_init()

vrna_ostream_t vrna_ostream_init (
            vrna_callback_stream_output * *output,*
            void * *auxdata* )

#include <ViennaRNA/stream_output.h>

Get an initialized ordered output stream.

**See also**

vrna_ostream_free(), vrna_ostream_request(), vrna_ostream_provide()

**Parameters**

| output | A callback function that processes and releases data in the stream |
|--------|-------------------------------------------------------------------|
| auxdata | A pointer to auxiliary data passed as first argument to the `output` callback |

**Returns**

An initialized ordered output stream

### 15.62.3.2 vrna_ostream_free()

void vrna_ostream_free (
            vrna_ostream_t *dat* )

#include <ViennaRNA/stream_output.h>

Free an initialized ordered output stream.

**See also**

vrna_ostream_init()

**Parameters**

| | |
|---|---|
| *dat* | The output stream for which occupied memory should be free'd |

**15.62.3.3   vrna_ostream_request()**

```
void vrna_ostream_request (
            vrna_ostream_t dat,
            unsigned int num )
```

#include <ViennaRNA/stream_output.h>

Request index in ordered output stream.

This function must be called prior to vrna_ostream_provide() to indicate that data associated with a certain index number is expected to be inserted into the stream in the future.

**See also**

vrna_ostream_init(), vrna_ostream_provide(), vrna_ostream_free()

dat The output stream for which the index is requested

**Parameters**

| | |
|---|---|
| *num* | The index to request data for |

**15.62.3.4   vrna_ostream_provide()**

```
void vrna_ostream_provide (
            vrna_ostream_t dat,
            unsigned int i,
            void * data )
```

#include <ViennaRNA/stream_output.h>

Provide output stream data for a particular index.

**Precondition**

The index data is provided for must have been requested using vrna_ostream_request() beforehand.

**See also**

vrna_ostream_request()

**Parameters**

| | |
|---|---|
| *dat* | The output stream for which data is provided |
| *i* | The index of the provided data |
| *data* | The data provided |

## 15.63 Functions for Creating Secondary Structure Plots, Dot-Plots, and More

### 15.63.1 Detailed Description

Collaboration diagram for Functions for Creating Secondary Structure Plots, Dot-Plots, and More:

### Modules

- Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others

### Files

- file naview.h
- file plot_aln.h

    *Various functions for plotting Sequence / Structure Alignments.*
- file plot_layouts.h

    *Secondary structure plot layout algorithms.*
- file plot_structure.h

    *Various functions for plotting RNA secondary structures.*
- file plot_utils.h

    *Various utilities to assist in plotting secondary structures and consensus structures.*
- file PS_dot.h

    *Various functions for plotting RNA secondary structures, dot-plots and other visualizations.*

### Data Structures

- struct COORDINATE

    *this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE More...*
- struct vrna_dotplot_auxdata_t

### Macros

- #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 PS_color_aln (const char ∗structure, const char ∗filename, const char ∗seqs[ ], const char ∗names[ ])

    *Produce PostScript sequence alignment color-annotated by consensus structure.*

- int aliPS_color_aln (const char ∗structure, const char ∗filename, const char ∗seqs[ ], const char ∗names[ ])
- 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*

- 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 PS_dot_plot (char ∗string, char ∗file)

    *Produce postscript dot-plot.*

**Variables**

- int rna_plot_type

    *Switch for changing the secondary structure layout algorithm.*

## 15.63.2 Data Structure Documentation

### 15.63.2.1 struct COORDINATE

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

### 15.63.2.2 struct vrna_dotplot_auxdata_t

Collaboration diagram for vrna_dotplot_auxdata_t:

## 15.63.3 Macro Definition Documentation

### 15.63.3.1 VRNA_PLOT_TYPE_SIMPLE

```
#define VRNA_PLOT_TYPE_SIMPLE 0
```

```
#include <ViennaRNA/plot_layouts.h>
```

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, vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot(), svg_rna_plot(), gmlRNA(), ssv_rna_plot(), xrna_plot()

**15.63.3.2 VRNA_PLOT_TYPE_NAVIEW**

```
#define VRNA_PLOT_TYPE_NAVIEW 1
```

#include <ViennaRNA/plot_layouts.h>

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, vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot(), svg_rna_plot(), gmlRNA(), ssv_rna_plot(), xrna_plot()

**15.63.3.3 VRNA_PLOT_TYPE_CIRCULAR**

```
#define VRNA_PLOT_TYPE_CIRCULAR 2
```

#include <ViennaRNA/plot_layouts.h>

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, vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot(), svg_rna_plot(), gmlRNA(), ssv_rna_plot(), xrna_plot()

**15.63.4 Function Documentation**

**15.63.4.1 aliPS_color_aln()**

```
int aliPS_color_aln (
            const char * structure,
            const char * filename,
            const char * seqs[],
            const char * names[] )
```

#include <ViennaRNA/plot_aln.h>

PS_color_aln for duplexes

**15.63.4.2 simple_xy_coordinates()**

```
int simple_xy_coordinates (
            short * pair_table,
            float * X,
            float * Y )
```

#include <ViennaRNA/plot_layouts.h>

Calculate nucleotide coordinates for secondary structure plot the *Simple way*

**See also**

make_pair_table(), rna_plot_type, simple_circplot_coordinates(), naview_xy_coordinates(), vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot, 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

**15.63.4.3 simple_circplot_coordinates()**

```
int simple_circplot_coordinates (
            short * pair_table,
            float * x,
            float * y )
```

#include <ViennaRNA/plot_layouts.h>

Calculate nucleotide coordinates for *Circular Plot*

This function calculates the coordinates of nucleotides mapped in equal distances 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(), vrna_file_PS_rnaplot_a(), vrna_file_PS_rnaplot, 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

**15.63.4.4    PS_dot_plot_list()**

```
int PS_dot_plot_list (
              char * seq,
              char * filename,
              plist * pl,
              plist * mf,
              char * comment )
```

#include <ViennaRNA/PS_dot.h>

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

**15.63.4.5    PS_dot_plot()**

```
int PS_dot_plot (
              char * string,
              char * file )
```

#include <ViennaRNA/PS_dot.h>

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 THREADSAFE

**Deprecated** This function is deprecated and will be removed soon! Use PS_dot_plot_list() instead!

## 15.63.5 Variable Documentation

### 15.63.5.1 rna_plot_type

`int rna_plot_type`

`#include <ViennaRNA/plot_layouts.h>`

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 [3].

**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

## 15.64 Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others

### 15.64.1 Detailed Description

Collaboration diagram for Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others:

**Functions**

- char ∗∗ vrna_annotate_covar_struct (const char ∗∗alignment, const char ∗structure, vrna_md_t ∗md)

  *Produce covariance annotation for an alignment given a secondary structure.*
- vrna_cpair_t ∗ vrna_annotate_covar_pairs (const char ∗∗alignment, vrna_ep_t ∗pl, vrna_ep_t ∗mfel, double threshold, vrna_md_t ∗md)

  *Produce covariance annotation for an alignment given a set of base pairs.*

## 15.65 Functions to convert between various units

### 15.65.1 Detailed Description

Collaboration diagram for Functions to convert between various units:

**Files**

- file units.h

    *Physical Units and Functions to convert them into each other.*

**Enumerations**

- enum vrna_unit_energy_e {
    VRNA_UNIT_J, VRNA_UNIT_KJ, VRNA_UNIT_CAL_IT, VRNA_UNIT_DACAL_IT,
    VRNA_UNIT_KCAL_IT, VRNA_UNIT_CAL, VRNA_UNIT_DACAL, VRNA_UNIT_KCAL,
    VRNA_UNIT_G_TNT, VRNA_UNIT_KG_TNT, VRNA_UNIT_T_TNT, VRNA_UNIT_EV,
    VRNA_UNIT_WH, VRNA_UNIT_KWH }

    *Energy / Work Units.*
- enum vrna_unit_temperature_e {
    VRNA_UNIT_K, VRNA_UNIT_DEG_C, VRNA_UNIT_DEG_F, VRNA_UNIT_DEG_R,
    VRNA_UNIT_DEG_N, VRNA_UNIT_DEG_DE, VRNA_UNIT_DEG_RE, VRNA_UNIT_DEG_RO }

    *Temperature Units.*

**Functions**

- double vrna_convert_energy (double energy, vrna_unit_energy_e from, vrna_unit_energy_e to)

    *Convert between energy / work units.*
- double vrna_convert_temperature (double temp, vrna_unit_temperature_e from, vrna_unit_temperature_e to)

    *Convert between temperature units.*

### 15.65.2 Enumeration Type Documentation

#### 15.65.2.1 vrna_unit_energy_e

enum vrna_unit_energy_e

#include <ViennaRNA/units.h>

Energy / Work Units.

**See also**

vrna_convert_energy()

**Enumerator**

| | |
|---|---|
| VRNA_UNIT_J | Joule ( $1\ J = 1\ kg \cdot m^2 s^{-2}$ ) |
| VRNA_UNIT_KJ | Kilojoule ( $1\ kJ = 1,000\ J$ ) |
| VRNA_UNIT_CAL_IT | Calorie (International (Steam) Table, $1\ cal_{IT} = 4.1868\ J$ ) |
| VRNA_UNIT_DACAL_IT | Decacolorie (International (Steam) Table, $1\ dacal_{IT} = 10\ cal_{IT} = 41.868\ J$ ) |
| VRNA_UNIT_KCAL_IT | Kilocalorie (International (Steam) Table, $1\ kcal_{IT} = 4.1868\ kJ$ ) |
| VRNA_UNIT_CAL | Calorie (Thermochemical, $1\ cal_{th} = 4.184\ J$ ) |
| VRNA_UNIT_DACAL | Decacalorie (Thermochemical, $1\ dacal_{th} = 10\ cal_{th} = 41.84\ J$ ) |
| VRNA_UNIT_KCAL | Kilocalorie (Thermochemical, $1\ kcal_{th} = 4.184\ kJ$ ) |
| VRNA_UNIT_G_TNT | g TNT ( $1\ g\ \mathrm{TNT} = 1,000\ cal_{th} = 4,184\ J$ ) |
| VRNA_UNIT_KG_TNT | kg TNT ( $1\ kg\ \mathrm{TNT} = 1,000\ kcal_{th} = 4,184\ kJ$ ) |
| VRNA_UNIT_T_TNT | ton TNT ( $1\ t\ \mathrm{TNT} = 1,000,000\ kcal_{th} = 4,184\ MJ$ ) |
| VRNA_UNIT_EV | Electronvolt ( $1\ eV = 1.602176565 \times 10^{-19}\ J$ ) |
| VRNA_UNIT_WH | Watt hour ( $1\ W \cdot h = 1\ W \cdot 3,600s = 3,600\ J = 3.6\ kJ$ ) |
| VRNA_UNIT_KWH | Kilowatt hour ( $1\ kW \cdot h = 1\ kW \cdot 3,600\ s = 3,600\ kJ = 3.6\ MJ$ ) |

**15.65.2.2 vrna_unit_temperature_e**

enum vrna_unit_temperature_e

#include <ViennaRNA/units.h>

Temperature Units.

**See also**

> vrna_convert_temperature()

**Enumerator**

| | |
|---|---|
| VRNA_UNIT_K | Kelvin (K) |
| VRNA_UNIT_DEG_C | Degree Celcius (℃) ( $[^\circ C] = [K] - 273.15$ ) |
| VRNA_UNIT_DEG_F | Degree Fahrenheit (℉) ( $[^\circ F] = [K] \times \frac{9}{5} - 459.67$ ) |
| VRNA_UNIT_DEG_R | Degree Rankine (°R) ( $[^\circ R] = [K] \times \frac{9}{5}$ ) |
| VRNA_UNIT_DEG_N | Degree Newton (°N) ( $[^\circ N] = ([K] - 273.15) \times \frac{33}{100}$ ) |
| VRNA_UNIT_DEG_DE | Degree Delisle (°De) ( $[^\circ De] = (373.15 - [K])\frac{3}{2}$ ) |
| VRNA_UNIT_DEG_RE | Degree Réaumur (°Ré) ( $[^\circ Ré] = ([K] - 273.15) \times \frac{4}{5}$ ) |
| VRNA_UNIT_DEG_RO | Degree Rømer (°Rø) ( $[^\circ R] = ([K] - 273.15) \times \frac{21}{40} + 7.5$ ) |

**15.65.3   Function Documentation**

**15.65.3.1   vrna_convert_energy()**

```
double vrna_convert_energy (
            double energy,
            vrna_unit_energy_e from,
            vrna_unit_energy_e to )
```

#include <ViennaRNA/units.h>

Convert between energy / work units.

**See also**

vrna_unit_energy_e

**Parameters**

| energy | Input energy value |
|--------|--------------------|
| from   | Input unit         |
| to     | Output unit        |

**Returns**

Energy value in Output unit

**15.65.3.2   vrna_convert_temperature()**

```
double vrna_convert_temperature (
            double temp,
            vrna_unit_temperature_e from,
            vrna_unit_temperature_e to )
```

#include <ViennaRNA/units.h>

Convert between temperature units.

**See also**

vrna_unit_temperature_e

**Parameters**

| temp | Input temperature value |
|------|-------------------------|
| from | Input unit              |
| to   | Output unit             |

**Returns**

Temperature value in Output unit

# Chapter 16

# Data Structure Documentation

## 16.1 _struct_en Struct Reference

Data structure for [energy_of_move()](#)

### 16.1.1 Detailed Description

Data structure for [energy_of_move()](#)

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

- ViennaRNA/move_set.h

## 16.2 Postorder_list Struct Reference

Postorder data structure.

### 16.2.1 Detailed Description

Postorder data structure.

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

- ViennaRNA/[dist_vars.h](#)

## 16.3 swString Struct Reference

Some other data structure.

### 16.3.1 Detailed Description

Some other data structure.

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

- ViennaRNA/dist_vars.h

## 16.4 Tree Struct Reference

Tree data structure.

Collaboration diagram for Tree:

### 16.4.1 Detailed Description

Tree data structure.

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

- ViennaRNA/dist_vars.h

## 16.5 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.*

### 16.5.1 Detailed Description

Variables compound for 2Dfold partition function folding.

**Deprecated** This data structure will be removed from the library soon! Use vrna_fold_compound_t and the corresponding functions vrna_fold_compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

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

- ViennaRNA/2Dpfold.h

## 16.6 vrna_dimer_conc_s Struct Reference

Data structure for concentration dependency computations.

**Data Fields**

- double Ac_start

    *start concentration A*
- double Bc_start

    *start concentration B*
- double ABc

    *End concentration AB.*

### 16.6.1 Detailed Description

Data structure for concentration dependency computations.

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

- ViennaRNA/concentrations.h

## 16.7  vrna_hc_bp_storage_t Struct Reference

A base pair hard constraint.

### 16.7.1  Detailed Description

A base pair hard constraint.

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

- ViennaRNA/constraints_hard.h

## 16.8  vrna_sc_bp_storage_t Struct Reference

A base pair constraint.

### 16.8.1  Detailed Description

A base pair constraint.

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

- ViennaRNA/constraints_soft.h

## 16.9  vrna_sc_motif_s Struct Reference

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

- ViennaRNA/constraints_ligand.h

## 16.10  vrna_sequence_s Struct Reference

Data structure representing a nucleotide sequence.

**Data Fields**

- vrna_seq_type_e type
    *The type of sequence.*
- char ∗ string
    *The string representation of the sequence.*
- short ∗ encoding
    *The integer representation of the sequence.*
- unsigned int length
    *The length of the sequence.*

### 16.10.1 Detailed Description

Data structure representing a nucleotide sequence.

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

- ViennaRNA/sequence.h

## 16.11 vrna_structured_domains_s Struct Reference

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

- ViennaRNA/structured_domains.h

## 16.12 vrna_subopt_sol_s 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.*

### 16.12.1 Detailed Description

Solution element from subopt.c.

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

- ViennaRNA/subopt.h

## 16.13 vrna_unstructured_domain_motif_s Struct Reference

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

- ViennaRNA/unstructured_domains.h

# Chapter 17

# File Documentation

## 17.1  ViennaRNA/1.8.4_epars.h File Reference

Free energy parameters for parameter file conversion.

### 17.1.1  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

## 17.2  ViennaRNA/1.8.4_intloops.h File Reference

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

### 17.2.1 Detailed Description

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

## 17.3 ViennaRNA/2Dfold.h File Reference

MFE structures for base pair distance classes.

Include dependency graph for 2Dfold.h:

**Data Structures**

- struct vrna_sol_TwoD_t

    *Solution element returned from vrna_mfe_TwoD() More...*

- struct TwoDfold_vars

    *Variables compound for 2Dfold MFE folding. More...*

**Typedefs**

- typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t

    *Solution element returned from vrna_mfe_TwoD()*

- typedef struct TwoDfold_vars TwoDfold_vars

    *Variables compound for 2Dfold MFE folding.*

**Functions**

- vrna_sol_TwoD_t ∗ vrna_mfe_TwoD (vrna_fold_compound_t ∗vc, int distance1, int distance2)

    *Compute MFE's and representative for distance partitioning.*

- char ∗ vrna_backtrack5_TwoD (vrna_fold_compound_t ∗vc, int k, int l, unsigned int j)

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

- 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.*

- vrna_sol_TwoD_t ∗ 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.*

### 17.3.1 Detailed Description

MFE structures for base pair distance classes.

## 17.4 ViennaRNA/2Dpfold.h File Reference

Partition function implementations for base pair distance classes.

Include dependency graph for 2Dpfold.h:

### Data Structures

- struct vrna_sol_TwoD_pf_t

    *Solution element returned from vrna_pf_TwoD() More...*
- struct TwoDpfold_vars

    *Variables compound for 2Dfold partition function folding.*

### Typedefs

- typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t

    *Solution element returned from vrna_pf_TwoD()*

### Functions

- vrna_sol_TwoD_pf_t ∗ vrna_pf_TwoD (vrna_fold_compound_t ∗vc, int maxDistance1, int maxDistance2)

    *Compute the partition function for all distance classes.*
- char ∗ vrna_pbacktrack_TwoD (vrna_fold_compound_t ∗vc, int d1, int d2)

    *Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- char ∗ vrna_pbacktrack5_TwoD (vrna_fold_compound_t ∗vc, 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.*
- 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.*
- void destroy_TwoDpfold_variables (TwoDpfold_vars ∗vars)

    *Free all memory occupied by a TwoDpfold_vars datastructure.*
- vrna_sol_TwoD_pf_t ∗ 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.*

### 17.4.1 Detailed Description

Partition function implementations for base pair distance classes.

## 17.4.2 Function Documentation

### 17.4.2.1 get_TwoDpfold_variables()

```
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 TwoDpfold() . 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.

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

**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

### 17.4.2.2 destroy_TwoDpfold_variables()

```
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_variabless() or get↩ _TwoDpfold_variables_from_MFE()

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩ compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

**See also**

get_TwoDpfold_variables(), get_TwoDpfold_variables_from_MFE()

**Parameters**

| | |
|---|---|
| *vars* | the datastructure to be free'd |

### 17.4.2.3 TwoDpfoldList()

vrna_sol_TwoD_pf_t* 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 TwoDfold() 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

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩
        compound_TwoD(), vrna_pf_TwoD(), and vrna_fold_compound_free() instead!

**See also**

        get_TwoDpfold_variables(), destroy_TwoDpfold_variables(), vrna_sol_TwoD_pf_t

**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

### 17.4.2.4 TwoDpfold_pbacktrack()

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 TwoDpfold() preceding this function is mandatory!

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold↩
_compound_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack_TwoD(), and vrna_fold_compound_free() in-
stead!

**See also**

> TwoDpfold()

**Parameters**

| in | *vars* | the datastructure containing all necessary folding attributes and matrices |
|----|--------|------------------------------------------------------------------------------|
| in | *d1*   | the distance to reference1 (may be -1)                                        |
| in | *d2*   | the distance to reference2                                                    |

**Returns**

> A sampled secondary structure in dot-bracket notation

**17.4.2.5 TwoDpfold_pbacktrack5()**

```
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 TwoDpfold() preceding this function is mandatory!

**Deprecated** Use the new API that relies on vrna_fold_compound_t and the corresponding functions vrna_fold_↩
compound_TwoD(), vrna_pf_TwoD(), vrna_pbacktrack5_TwoD(), and vrna_fold_compound_free() instead!

**See also**

TwoDpfold_pbacktrack(), TwoDpfold()

**Parameters**

| in | *vars* | the datastructure containing all necessary folding attributes and matrices |
|----|--------|-----------------------------------------------------------------------------|
| in | *d1*   | the distance to reference1 (may be -1)                                       |
| in | *d2*   | the distance to reference2                                                   |
| in | *length* | the length of the structure beginning from the 5' end                     |

**Returns**

A sampled secondary structure in dot-bracket notation

## 17.5 ViennaRNA/alifold.h File Reference

Functions for comparative structure prediction using RNA sequence alignments.

Include dependency graph for alifold.h:

**Functions**

- float vrna_alifold (const char ∗∗sequences, char ∗structure)

  *Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.*

- float vrna_circalifold (const char ∗∗sequences, char ∗structure)

  *Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.*

- float vrna_pf_alifold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)

  *Compute Partition function $Q$ (and base pair probabilities) for an RNA sequence alignment using a comparative method.*

- float vrna_pf_circalifold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)

  *Compute Partition function $Q$ (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.*

- 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.*

- 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.*

- float alipf_fold_par (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl, vrna_exp_param_t ∗parameters, int calculate_bppm, int is_constrained, int is_circular)

- float alipf_fold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗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 vrna_pinfo_t structs. The list is terminated by the first entry with pi.i = 0.*

- float alipf_circ_fold (const char ∗∗sequences, char ∗structure, vrna_ep_t ∗∗pl)

- FLT_OR_DBL ∗ export_ali_bppm (void)

  *Get a pointer to the base pair probability array.*

- void free_alipf_arrays (void)

  *Free the memory occupied by folding matrices allocated by alipf_fold, alipf_circ_fold, etc.*

- 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, FLT_OR_DBL ∗∗qb_p, FLT_OR_DBL ∗∗qm_p, FLT_OR_DBL ∗∗q1k_p, FLT_OR_DBL ∗∗qln_p, short ∗∗pscore)

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

- void update_alifold_params (void)

  *Update the energy parameters for alifold function.*

## 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.*

### 17.5.1 Detailed Description

Functions for comparative structure prediction using RNA sequence alignments.

## 17.6 ViennaRNA/aln_util.h File Reference

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

Include dependency graph for aln_util.h:

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

**Data Structures**

- struct vrna_pinfo_s

    *A base pair info structure. More...*

**Macros**

- #define VRNA_MEASURE_SHANNON_ENTROPY 1U

    *Flag indicating Shannon Entropy measure.*

**Typedefs**

- typedef struct vrna_pinfo_s vrna_pinfo_t

    *Typename for the base pair info repesenting data structure vrna_pinfo_s.*

- typedef struct vrna_pinfo_s pair_info

    *Old typename of vrna_pinfo_s.*

**Functions**

- int vrna_aln_mpi (const char ∗∗alignment)

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

- vrna_pinfo_t ∗ vrna_aln_pinfo (vrna_fold_compound_t ∗vc, const char ∗structure, double threshold)

    *Retrieve an array of vrna_pinfo_t structures from precomputed pair probabilities.*

- char ∗∗ vrna_aln_slice (const char ∗∗alignment, unsigned int i, unsigned int j)

    *Slice out a subalignment from a larger alignment.*

- void vrna_aln_free (char ∗∗alignment)

    *Free memory occupied by a set of aligned sequences.*

- char ∗∗ vrna_aln_uppercase (const char ∗∗alignment)

    *Create a copy of an alignment with only uppercase letters in the sequences.*

- char ∗∗ vrna_aln_toRNA (const char ∗∗alignment)

    *Create a copy of an alignment where DNA alphabet is replaced by RNA alphabet.*

- char ∗∗ vrna_aln_copy (const char ∗∗alignment, unsigned int options)

    *Make a copy of a multiple sequence alignment.*

- float ∗ vrna_aln_conservation_struct (const char ∗∗alignment, const char ∗structure, const vrna_md_t ∗md)

    *Compute base pair conservation of a consensus structure.*

- float ∗ vrna_aln_conservation_col (const char ∗∗alignment, const vrna_md_t ∗md_p, unsigned int options)

    *Compute nucleotide conservation in an alignment.*

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

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

- 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.*

### 17.6.1 Detailed Description

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

## 17.7 ViennaRNA/alphabet.h File Reference

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

Include dependency graph for alphabet.h:

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

**Functions**

- char ∗ vrna_ptypes (const short ∗S, vrna_md_t ∗md)

  *Get an array of the numerical encoding for each possible base pair (i,j)*
- short ∗ vrna_seq_encode (const char ∗sequence, vrna_md_t ∗md)

  *Get a numerical representation of the nucleotide sequence.*
- short ∗ vrna_seq_encode_simple (const char ∗sequence, vrna_md_t ∗md)

  *Get a numerical representation of the nucleotide sequence (simple version)*
- int vrna_nucleotide_encode (char c, vrna_md_t ∗md)

  *Encode a nucleotide character to numerical value.*
- char vrna_nucleotide_decode (int enc, vrna_md_t ∗md)

  *Decode a numerical representation of a nucleotide back into nucleotide alphabet.*

### 17.7.1 Detailed Description

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

## 17.8 ViennaRNA/boltzmann_sampling.h File Reference

Boltzmann Sampling of secondary structures from the ensemble.

Include dependency graph for boltzmann_sampling.h:

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

**Functions**

- char ∗ vrna_pbacktrack5 (vrna_fold_compound_t ∗vc, int length)

    *Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- char ∗ vrna_pbacktrack (vrna_fold_compound_t ∗vc)

    *Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.*

### 17.8.1 Detailed Description

Boltzmann Sampling of secondary structures from the ensemble.

A.k.a. Stochastic backtracking

## 17.9 ViennaRNA/centroid.h File Reference

Centroid structure computation.

Include dependency graph for centroid.h:

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

**Functions**

- char ∗ vrna_centroid (vrna_fold_compound_t ∗vc, double ∗dist)

    *Get the centroid structure of the ensemble.*
- char ∗ vrna_centroid_from_plist (int length, double ∗dist, vrna_ep_t ∗pl)

    *Get the centroid structure of the ensemble.*
- char ∗ vrna_centroid_from_probs (int length, double ∗dist, FLT_OR_DBL ∗probs)

    *Get the centroid structure of the ensemble.*
- char ∗ get_centroid_struct_pl (int length, double ∗dist, vrna_ep_t ∗pl)

    *Get the centroid structure of the ensemble.*
- char ∗ get_centroid_struct_pr (int length, double ∗dist, FLT_OR_DBL ∗pr)

    *Get the centroid structure of the ensemble.*

### 17.9.1 Detailed Description

Centroid structure computation.

### 17.9.2 Function Documentation

**17.9.2.1 get_centroid_struct_pl()**

```
char* get_centroid_struct_pl (
            int length,
            double * dist,
            vrna_ep_t * pl )
```

Get the centroid structure of the ensemble.

**Deprecated** This function was renamed to vrna_centroid_from_plist()

**17.9.2.2 get_centroid_struct_pr()**

```
char* get_centroid_struct_pr (
            int length,
            double * dist,
            FLT_OR_DBL * pr )
```

Get the centroid structure of the ensemble.

**Deprecated** This function was renamed to vrna_centroid_from_probs()

## 17.10 ViennaRNA/char_stream.h File Reference

Implementation of a dynamic, buffered character stream.

Include dependency graph for char_stream.h:

### 17.10.1 Detailed Description

Implementation of a dynamic, buffered character stream.

## 17.11 ViennaRNA/cofold.h File Reference

MFE implementations for RNA-RNA interaction.

Include dependency graph for cofold.h:

**Functions**

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

    *Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.*
- 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, vrna_param_t ∗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 update_cofold_params_par (vrna_param_t ∗parameters)

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

    *get_monomer_free_energies*
- void initialize_cofold (int length)

### 17.11.1 Detailed Description

MFE implementations for RNA-RNA interaction.

## 17.12 ViennaRNA/combinatorics.h File Reference

Various implementations that deal with combinatorial aspects of RNA/DNA folding.

**Functions**

- unsigned int ∗∗ vrna_enumerate_necklaces (const unsigned int ∗type_counts)

    *Enumerate all necklaces with fixed content.*

### 17.12.1 Detailed Description

Various implementations that deal with combinatorial aspects of RNA/DNA folding.

## 17.13 ViennaRNA/commands.h File Reference

Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.

Include dependency graph for commands.h:

**Data Structures**

- struct vrna_command_s

    *List element for commands ready for application to a vrna_fold_compound_t. More...*

**Macros**

- #define VRNA_CMD_PARSE_HC 1U

    *Command parse/apply flag indicating hard constraints.*
- #define VRNA_CMD_PARSE_SC 2U

    *Command parse/apply flag indicating soft constraints.*
- #define VRNA_CMD_PARSE_UD 4U

    *Command parse/apply flag indicating unstructured domains.*
- #define VRNA_CMD_PARSE_SD 8U

    *Command parse/apply flag indicating structured domains.*
- #define VRNA_CMD_PARSE_DEFAULTS

    *Command parse/apply flag indicating default set of commands.*

**Typedefs**

- typedef struct vrna_command_s vrna_cmd_t

    *Typename for the command repesenting data structure vrna_command_s.*

**Enumerations**

- enum vrna_command_e

    *Types of commands within a list of vrna_command_s structures.*

**Functions**

- vrna_cmd_t ∗ vrna_file_commands_read (const char ∗filename, unsigned int options)

    *Extract a list of commands from a command file.*
- int vrna_file_commands_apply (vrna_fold_compound_t ∗vc, const char ∗filename, unsigned int options)

    *Apply a list of commands from a command file.*
- int vrna_commands_apply (vrna_fold_compound_t ∗vc, vrna_cmd_t ∗commands, unsigned int options)

    *Apply a list of commands to a vrna_fold_compound_t.*
- void vrna_commands_free (vrna_cmd_t ∗commands)

    *Free memory occupied by a list of commands.*

**17.13.1  Detailed Description**

Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.

## 17.14 ViennaRNA/concentrations.h File Reference

Concentration computations for RNA-RNA interactions.

Include dependency graph for concentrations.h:

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

### Data Structures

- struct vrna_dimer_conc_s

    *Data structure for concentration dependency computations.*

### Functions

- vrna_dimer_conc_t ∗ 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.*

- typedef struct vrna_dimer_conc_s vrna_dimer_conc_t

    *Typename for the data structure that stores the dimer concentrations, vrna_dimer_conc_s, as required by vrna_pf↩ _dimer_concentration()*

- typedef struct vrna_dimer_conc_s ConcEnt

    *Backward compatibility typedef for vrna_dimer_conc_s.*

- vrna_dimer_conc_t ∗ vrna_pf_dimer_concentrations (double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double ∗startconc, const vrna_exp_param_t ∗exp_params)

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

### 17.14.1 Detailed Description

Concentration computations for RNA-RNA interactions.

### 17.14.2 Function Documentation

**17.14.2.1 get_concentrations()**

vrna_dimer_conc_t* 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 vrna_dimer_pf_t struct.

**Deprecated** { Use vrna_pf_dimer_concentrations() instead!}

**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**

    vrna_dimer_conc_t array containing the equilibrium energies and start concentrations

## 17.15 ViennaRNA/constraints.h File Reference

Functions and data structures for constraining secondary structure predictions and evaluation.

Include dependency graph for constraints.h:

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

## Macros

- #define VRNA_CONSTRAINT_FILE 0

    *Flag for vrna_constraints_add() to indicate that constraints are present in a text file.*
- #define VRNA_CONSTRAINT_SOFT_MFE 0

    *Indicate generation of constraints for MFE folding.*
- #define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF

    *Indicate generation of constraints for partition function computation.*
- #define VRNA_DECOMP_PAIR_HP (unsigned char)1

    *Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.*
- #define VRNA_DECOMP_PAIR_IL (unsigned char)2

    *Indicator for interior loop decomposition step.*
- #define VRNA_DECOMP_PAIR_ML (unsigned char)3

    *Indicator for multibranch loop decomposition step.*
- #define VRNA_DECOMP_ML_ML_ML (unsigned char)5

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_STEM (unsigned char)6

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_ML (unsigned char)7

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_UP (unsigned char)8

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_ML_STEM (unsigned char)9

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_COAXIAL (unsigned char)10

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_ML_COAXIAL_ENC (unsigned char)11

    *Indicator for decomposition of multibranch loop part.*
- #define VRNA_DECOMP_EXT_EXT (unsigned char)12

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_UP (unsigned char)13

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_STEM (unsigned char)14

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_EXT_EXT (unsigned char)15

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_STEM_EXT (unsigned char)16

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_STEM_OUTSIDE (unsigned char)17

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_EXT_STEM (unsigned char)18

    *Indicator for decomposition of exterior loop part.*
- #define VRNA_DECOMP_EXT_EXT_STEM1 (unsigned char)19

    *Indicator for decomposition of exterior loop part.*

## Functions

- void vrna_constraints_add (vrna_fold_compound_t ∗vc, const char ∗constraint, unsigned int options)

    *Add constraints to a vrna_fold_compound_t data structure.*

### 17.15.1 Detailed Description

Functions and data structures for constraining secondary structure predictions and evaluation.

## 17.16 ViennaRNA/constraints_hard.h File Reference

Functions and data structures for handling of secondary structure hard constraints.

Include dependency graph for constraints_hard.h:

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

### Data Structures

- struct vrna_hc_bp_storage_t

    *A base pair hard constraint.*
- struct vrna_hc_s

    *The hard constraints data structure. More...*
- struct vrna_hc_up_s

    *A single hard constraint for a single nucleotide. More...*

### Macros

- #define VRNA_CONSTRAINT_NO_HEADER 0

    *do not print the header information line*
- #define VRNA_CONSTRAINT_DB 16384U

    *Flag for vrna_constraints_add() to indicate that constraint is passed in pseudo dot-bracket notation.*
- #define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U

    *Switch for dot-bracket structure constraint to enforce base pairs.*
- #define VRNA_CONSTRAINT_DB_PIPE 65536U

    *Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.*
- #define VRNA_CONSTRAINT_DB_DOT 131072U

    *dot '.' switch for structure constraints (no constraint at all)*
- #define VRNA_CONSTRAINT_DB_X 262144U

    *'x' switch for structure constraint (base must not pair)*
- #define VRNA_CONSTRAINT_DB_ANG_BRACK 524288U

    *angle brackets '$<$', '$>$' switch for structure constraint (paired downstream/upstream)*
- #define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U

    *round brackets '(',')' switch for structure constraint (base i pairs base j)*
- #define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U

    *Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.*
- #define VRNA_CONSTRAINT_DB_INTERMOL 4194304U

    *Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.*

- #define VRNA_CONSTRAINT_DB_GQUAD 8388608U

    *'+' switch for structure constraint (base is involved in a gquad)*
- #define VRNA_CONSTRAINT_DB_WUSS 33554432U

    *Flag to indicate Washington University Secondary Structure (WUSS) notation of the hard constraint string.*
- #define VRNA_CONSTRAINT_DB_DEFAULT

    *Switch for dot-bracket structure constraint with default symbols.*
- #define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP (unsigned char)0x01

    *Hard constraints flag, base pair in the exterior loop.*
- #define VRNA_CONSTRAINT_CONTEXT_HP_LOOP (unsigned char)0x02

    *Hard constraints flag, base pair encloses hairpin loop.*
- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP (unsigned char)0x04

    *Hard constraints flag, base pair encloses an interior loop.*
- #define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC (unsigned char)0x08

    *Hard constraints flag, base pair encloses a multi branch loop.*
- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP (unsigned char)0x10

    *Hard constraints flag, base pair is enclosed in an interior loop.*
- #define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC (unsigned char)0x20

    *Hard constraints flag, base pair is enclosed in a multi branch loop.*
- #define VRNA_CONSTRAINT_CONTEXT_ENFORCE (unsigned char)0x40

    *Hard constraint flag to indicate enforcement of constraints.*
- #define VRNA_CONSTRAINT_CONTEXT_NO_REMOVE (unsigned char)0x80

    *Hard constraint flag to indicate not to remove base pairs that conflict with a given constraint.*
- #define VRNA_CONSTRAINT_CONTEXT_NONE (unsigned char)0

    *Constraint context flag that forbids any loop.*
- #define VRNA_CONSTRAINT_CONTEXT_CLOSING_LOOPS

    *Constraint context flag indicating base pairs that close any loop.*
- #define VRNA_CONSTRAINT_CONTEXT_ENCLOSED_LOOPS

    *Constraint context flag indicating base pairs enclosed by any loop.*
- #define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS

    *Constraint context flag indicating any loop context.*

## Typedefs

- typedef struct vrna_hc_s vrna_hc_t

    *Typename for the hard constraints data structure vrna_hc_s.*
- typedef struct vrna_hc_up_s vrna_hc_up_t

    *Typename for the single nucleotide hard constraint data structure vrna_hc_up_s.*
- typedef unsigned char() vrna_callback_hc_evaluate(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.*

## Enumerations

- enum vrna_hc_type_e { VRNA_HC_DEFAULT, VRNA_HC_WINDOW }

    *The hard constraints type.*

**Functions**

- void vrna_message_constraint_options (unsigned int option)

  *Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)*
- void vrna_message_constraint_options_all (void)

  *Print structure constraint characters to stdout (full constraint support)*
- void vrna_hc_init (vrna_fold_compound_t *vc)

  *Initialize/Reset hard constraints to default values.*
- void vrna_hc_add_up (vrna_fold_compound_t *vc, int i, unsigned char option)

  *Make a certain nucleotide unpaired.*
- int vrna_hc_add_up_batch (vrna_fold_compound_t *vc, vrna_hc_up_t *constraints)

  *Apply a list of hard constraints for single nucleotides.*
- void vrna_hc_add_bp (vrna_fold_compound_t *vc, int i, int j, unsigned char option)

  *Favorize/Enforce a certain base pair (i,j)*
- void vrna_hc_add_bp_nonspecific (vrna_fold_compound_t *vc, int i, int d, unsigned char option)

  *Enforce a nucleotide to be paired (upstream/downstream)*
- void vrna_hc_free (vrna_hc_t *hc)

  *Free the memory allocated by a vrna_hc_t data structure.*
- void vrna_hc_add_f (vrna_fold_compound_t *vc, vrna_callback_hc_evaluate *f)

  *Add a function pointer pointer for the generic hard constraint feature.*
- void vrna_hc_add_data (vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *f)

  *Add an auxiliary data structure for the generic hard constraints callback function.*
- int vrna_hc_add_from_db (vrna_fold_compound_t *vc, const char *constraint, unsigned int options)

  *Add hard constraints from pseudo dot-bracket notation.*
- void print_tty_constraint (unsigned int option)

  *Print structure constraint characters to stdout. (constraint support is specified by option parameter)*
- void print_tty_constraint_full (void)

  *Print structure constraint characters to stdout (full constraint support)*
- 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.*

## 17.16.1 Detailed Description

Functions and data structures for handling of secondary structure hard constraints.

## 17.16.2 Macro Definition Documentation

### 17.16.2.1 VRNA_CONSTRAINT_NO_HEADER

```
#define VRNA_CONSTRAINT_NO_HEADER 0
```

do not print the header information line

**Deprecated** This mode is not supported anymore!

**17.16.2.2 VRNA_CONSTRAINT_DB_ANG_BRACK**

```
#define VRNA_CONSTRAINT_DB_ANG_BRACK 524288U
```

angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)

**See also**

vrna_hc_add_from_db(), vrna_constraints_add(), vrna_message_constraint_options(), vrna_message_constraint_options_all()

**17.16.3 Enumeration Type Documentation**

**17.16.3.1 vrna_hc_type_e**

```
enum vrna_hc_type_e
```

The hard constraints type.

Global and local structure prediction methods use a slightly different way to handle hard constraints internally. This enum is used to distinguish both types.

**Enumerator**

| VRNA_HC_DEFAULT | Default Hard Constraints. |
| --- | --- |
| VRNA_HC_WINDOW | Hard Constraints suitable for local structure prediction using window approach. |
| | **See also** |
| | vrna_mfe_window(), vrna_mfe_window_zscore(), pfl_fold() |

**17.16.4 Function Documentation**

**17.16.4.1 vrna_hc_add_data()**

```
void vrna_hc_add_data (
            vrna_fold_compound_t * vc,
            void * data,
            vrna_callback_free_auxdata * f )
```

Add an auxiliary data structure for the generic hard constraints callback function.

**See also**

vrna_hc_add_f()

---

**Parameters**

| | |
|---|---|
| *vc* | The fold compound the generic hard constraint function should be bound to |
| *data* | A pointer to the data structure that holds required data for function 'f' |
| *free_data* | A pointer to a function that free's the memory occupied by (Maybe NULL) |

**17.16.4.2 print_tty_constraint()**

```
void print_tty_constraint (
              unsigned int option )
```

Print structure constraint characters to stdout. (constraint support is specified by option parameter)

**Deprecated** Use vrna_message_constraints() instead!

**Parameters**

| | |
|---|---|
| *option* | Option switch that tells which constraint help will be printed |

**17.16.4.3 print_tty_constraint_full()**

```
void print_tty_constraint_full (
              void )
```

Print structure constraint characters to stdout (full constraint support)

**Deprecated** Use vrna_message_constraint_options_all() instead!

**17.16.4.4 constrain_ptypes()**

```
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.

**Deprecated** Do not use this function anymore! Structure constraints are now handled through vrna_hc_t and related functions.

**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 |
| *BP* | (not used anymore) |
| *min_loop_size* | The minimal loop size (usually TURN ) |
| *idx_type* | Define the access type for base pair type array (0 = indx, 1 = iindx) |

## 17.17 ViennaRNA/constraints_ligand.h File Reference

Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.

Include dependency graph for constraints_ligand.h:

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

**Data Structures**

- struct vrna_sc_motif_s

**Functions**

- int vrna_sc_add_hi_motif (vrna_fold_compound_t ∗vc, const char ∗seq, const char ∗structure, FLT_OR_DBL energy, unsigned int options)

    *Add soft constraints for hairpin or interior loop binding motif.*

### 17.17.1 Detailed Description

Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.

## 17.18 ViennaRNA/constraints_SHAPE.h File Reference

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

Include dependency graph for constraints_SHAPE.h:

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

**Functions**

- int vrna_sc_add_SHAPE_deigan (vrna_fold_compound_t ∗vc, const double ∗reactivities, double m, double b, unsigned int options)

    *Add SHAPE reactivity data as soft constraints (Deigan et al. method)*

- int vrna_sc_add_SHAPE_deigan_ali (vrna_fold_compound_t ∗vc, const char ∗∗shape_files, const int ∗shape_file_association, double m, double b, unsigned int options)

    *Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)*

- int vrna_sc_add_SHAPE_zarringhalam (vrna_fold_compound_t ∗vc, const double ∗reactivities, double b, double default_value, const char ∗shape_conversion, unsigned int options)

    *Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)*

- int vrna_sc_SHAPE_parse_method (const char ∗method_string, char ∗method, float ∗param_1, float ∗param_2)

    *Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.*

- int vrna_sc_SHAPE_to_pr (const char ∗shape_conversion, double ∗values, int length, double default_value)

    *Convert SHAPE reactivity values to probabilities for being unpaired.*

## 17.18.1 Detailed Description

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

## 17.18.2 Function Documentation

### 17.18.2.1 vrna_sc_SHAPE_parse_method()

```
int vrna_sc_SHAPE_parse_method (
            const char * method_string,
            char * method,
            float * param_1,
            float * param_2 )
```

Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.

**Parameters**

| | |
|---|---|
| *method_string* | The string that contains the encoded SHAPE reactivity conversion method |
| *method* | A pointer to the memory location where the method character will be stored |
| *param_1* | A pointer to the memory location where the first parameter of the corresponding method will be stored |
| *param_2* | A pointer to the memory location where the second parameter of the corresponding method will be stored |

**Returns**

> 1 on successful extraction of the method, 0 on errors

## 17.19 ViennaRNA/constraints_soft.h File Reference

Functions and data structures for secondary structure soft constraints.

Include dependency graph for constraints_soft.h:

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

### Data Structures

- struct vrna_sc_bp_storage_t

    *A base pair constraint.*
- struct vrna_sc_s

    *The soft constraints data structure. More...*

### Typedefs

- typedef struct vrna_sc_s vrna_sc_t

    *Typename for the soft constraints data structure vrna_sc_s.*
- typedef int() vrna_callback_sc_energy(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to retrieve pseudo energy contribution for soft constraint feature.*
- typedef FLT_OR_DBL() vrna_callback_sc_exp_energy(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.*
- typedef vrna_basepair_t ∗() vrna_callback_sc_backtrack(int i, int j, int k, int l, unsigned char d, void ∗data)

    *Callback to retrieve auxiliary base pairs for soft constraint feature.*

### Enumerations

- enum vrna_sc_type_e { VRNA_SC_DEFAULT, VRNA_SC_WINDOW }

**Functions**

- void vrna_sc_init (vrna_fold_compound_t ∗vc)

    *Initialize an empty soft constraints data structure within a vrna_fold_compound_t.*
- void vrna_sc_set_bp (vrna_fold_compound_t ∗vc, const FLT_OR_DBL ∗∗constraints, unsigned int options)

    *Set soft constraints for paired nucleotides.*
- void vrna_sc_add_bp (vrna_fold_compound_t ∗vc, int i, int j, FLT_OR_DBL energy, unsigned int options)

    *Add soft constraints for paired nucleotides.*
- void vrna_sc_set_up (vrna_fold_compound_t ∗vc, const FLT_OR_DBL ∗constraints, unsigned int options)

    *Set soft constraints for unpaired nucleotides.*
- void vrna_sc_add_up (vrna_fold_compound_t ∗vc, int i, FLT_OR_DBL energy, unsigned int options)

    *Add soft constraints for unpaired nucleotides.*
- void vrna_sc_remove (vrna_fold_compound_t ∗vc)

    *Remove soft constraints from vrna_fold_compound_t.*
- void vrna_sc_free (vrna_sc_t ∗sc)

    *Free memory occupied by a vrna_sc_t data structure.*
- void vrna_sc_add_data (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗free_data)

    *Add an auxiliary data structure for the generic soft constraints callback function.*
- void vrna_sc_add_f (vrna_fold_compound_t ∗vc, vrna_callback_sc_energy ∗f)

    *Bind a function pointer for generic soft constraint feature (MFE version)*
- void vrna_sc_add_bt (vrna_fold_compound_t ∗vc, vrna_callback_sc_backtrack ∗f)

    *Bind a backtracking function pointer for generic soft constraint feature.*
- void vrna_sc_add_exp_f (vrna_fold_compound_t ∗vc, vrna_callback_sc_exp_energy ∗exp_f)

    *Bind a function pointer for generic soft constraint feature (PF version)*

## 17.19.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

## 17.19.2 Enumeration Type Documentation

### 17.19.2.1 vrna_sc_type_e

enum vrna_sc_type_e

**Enumerator**

| VRNA_SC_DEFAULT | Default Soft Constraints. |
|---|---|
| VRNA_SC_WINDOW | Soft Constraints suitable for local structure prediction using window approach. |
| | **See also** |
| | vrna_mfe_window(), vrna_mfe_window_zscore(), pfl_fold() |

## 17.20 ViennaRNA/convert_epars.h File Reference

Functions and definitions for energy parameter file format conversion.

### Macros

- #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)

### 17.20.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

## 17.21 ViennaRNA/data_structures.h File Reference

Various data structures and pre-processor macros.

Include dependency graph for data_structures.h:

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

**Data Structures**

- struct vrna_basepair_s

  *Base pair data structure used in subopt.c. More...*
- struct vrna_cpair_s

  *this datastructure is used as input parameter in functions of PS_dot.c More...*
- struct vrna_color_s
- struct vrna_data_linear_s
- struct vrna_sect_s

  *Stack of partial structures for backtracking. More...*
- struct vrna_bp_stack_s

  *Base pair stack element. More...*
- struct pu_contrib

  *contributions to p_u More...*
- struct interact

  *interaction data structure for RNAup More...*
- struct pu_out

  *Collection of all free_energy of beeing unpaired values for output. More...*
- struct constrain

  *constraints for cofolding More...*
- struct duplexT

  *Data structure for RNAduplex. More...*
- struct node

  *Data structure for RNAsnoop (fold energy list) More...*
- struct snoopT

  *Data structure for RNAsnoop. More...*
- struct dupVar

  *Data structure used in RNApkplex. More...*

**Typedefs**

- typedef struct vrna_basepair_s vrna_basepair_t

  *Typename for the base pair repesenting data structure vrna_basepair_s.*
- typedef struct vrna_elem_prob_s vrna_plist_t

  *Typename for the base pair list repesenting data structure vrna_elem_prob_s.*
- typedef struct vrna_bp_stack_s vrna_bp_stack_t

  *Typename for the base pair stack repesenting data structure vrna_bp_stack_s.*
- typedef struct vrna_cpair_s vrna_cpair_t

  *Typename for data structure vrna_cpair_s.*
- typedef struct vrna_sect_s vrna_sect_t

  *Typename for stack of partial structures vrna_sect_s.*
- typedef double FLT_OR_DBL

  *Typename for floating point number in partition function computations.*
- typedef struct vrna_basepair_s PAIR

  *Old typename of vrna_basepair_s.*
- typedef struct vrna_elem_prob_s plist

  *Old typename of vrna_elem_prob_s.*
- typedef struct vrna_cpair_s cpair

  *Old typename of vrna_cpair_s.*
- typedef struct vrna_sect_s sect

*Old typename of vrna_sect_s.*

- typedef struct vrna_bp_stack_s bondT

    *Old typename of vrna_bp_stack_s.*

- typedef struct pu_contrib pu_contrib

    *contributions to p_u*

- typedef struct interact interact

    *interaction data structure for RNAup*

- typedef struct pu_out pu_out

    *Collection of all free_energy of beeing unpaired values for output.*

- typedef struct constrain constrain

    *constraints for cofolding*

- typedef struct node folden

    *Data structure for RNAsnoop (fold energy list)*

- typedef struct dupVar dupVar

    *Data structure used in RNApkplex.*

## Functions

- void vrna_C11_features (void)

    *Dummy symbol to check whether the library was build using C11/C++11 features.*

### 17.21.1 Detailed Description

Various data structures and pre-processor macros.

## 17.22 ViennaRNA/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

    *Postorder data structure.*

- struct Tree

    *Tree data structure.*

- struct swString

    *Some other data structure.*

**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.*

## 17.22.1 Detailed Description

Global variables for Distance-Package.

## 17.22.2 Variable Documentation

### 17.22.2.1 edit_backtrack

```
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

### 17.22.2.2 cost_matrix

```
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).

## 17.23 ViennaRNA/dp_matrices.h File Reference

Functions to deal with standard dynamic programming (DP) matrices.

Include dependency graph for dp_matrices.h:

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

**Data Structures**

- struct vrna_mx_mfe_s

  *Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...*

- struct vrna_mx_pf_s

  *Partition function (PF) Dynamic Programming (DP) matrices data structure required within the vrna_fold_compound_t. More...*

**Typedefs**

- typedef struct vrna_mx_mfe_s vrna_mx_mfe_t

  *Typename for the Minimum Free Energy (MFE) DP matrices data structure vrna_mx_mfe_s.*

- typedef struct vrna_mx_pf_s vrna_mx_pf_t

  *Typename for the Partition Function (PF) DP matrices data structure vrna_mx_pf_s.*

**Enumerations**

- enum vrna_mx_type_e { VRNA_MX_DEFAULT, VRNA_MX_WINDOW, VRNA_MX_2DFOLD }

  *An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.*

**Functions**

- int vrna_mx_add (vrna_fold_compound_t ∗vc, vrna_mx_type_e type, unsigned int options)

  *Add Dynamic Programming (DP) matrices (allocate memory)*

- void vrna_mx_mfe_free (vrna_fold_compound_t ∗vc)

  *Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.*

- void vrna_mx_pf_free (vrna_fold_compound_t ∗vc)

  *Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.*

### 17.23.1 Detailed Description

Functions to deal with standard dynamic programming (DP) matrices.

## 17.24 ViennaRNA/duplex.h File Reference

Functions for simple RNA-RNA duplex interactions.

Include dependency graph for duplex.h:

### 17.24.1 Detailed Description

Functions for simple RNA-RNA duplex interactions.

## 17.25 ViennaRNA/edit_cost.h File Reference

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

### 17.25.1 Detailed Description

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

## 17.26 ViennaRNA/energy_const.h File Reference

Energy parameter constants.

Include dependency graph for energy_const.h:

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

**Macros**

- #define GASCONST 1.98717 /∗ in [cal/K] ∗/
- #define K0 273.15
- #define INF 10000000 /∗ (INT_MAX/10) ∗/
- #define FORBIDDEN 9999
- #define BONUS 10000
- #define NBPAIRS 7
- #define TURN 3
- #define MAXLOOP 30

### 17.26.1 Detailed Description

Energy parameter constants.

### 17.26.2 Macro Definition Documentation

#### 17.26.2.1 GASCONST

```
#define GASCONST 1.98717 /* in [cal/K] */
```

The gas constant

---

**17.26.2.2 K0**

```
#define K0 273.15
```

0 deg Celsius in Kelvin

**17.26.2.3 INF**

```
#define INF 10000000 /* (INT_MAX/10) */
```

Infinity as used in minimization routines

**17.26.2.4 FORBIDDEN**

```
#define FORBIDDEN 9999
```

forbidden

**17.26.2.5 BONUS**

```
#define BONUS 10000
```

bonus contribution

**17.26.2.6 NBPAIRS**

```
#define NBPAIRS 7
```

The number of distinguishable base pairs

**17.26.2.7 TURN**

```
#define TURN 3
```

The minimum loop length

**17.26.2.8 MAXLOOP**

```
#define MAXLOOP 30
```

The maximum loop length

## 17.27 ViennaRNA/equilibrium_probs.h File Reference

Equilibrium Probability implementations.

Include dependency graph for equilibrium_probs.h:

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

**Functions**

- double vrna_mean_bp_distance_pr (int length, FLT_OR_DBL *pr)

  *Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.*

- double vrna_mean_bp_distance (vrna_fold_compound_t *vc)

  *Get the mean base pair distance in the thermodynamic ensemble.*

- vrna_ep_t * vrna_stack_prob (vrna_fold_compound_t *vc, double cutoff)

  *Compute stacking probabilities.*

- void vrna_pf_dimer_probs (double FAB, double FA, double FB, vrna_ep_t *prAB, const vrna_ep_t *prA, const vrna_ep_t *prB, int Alength, const vrna_exp_param_t *exp_params)

  *Compute Boltzmann probabilities of dimerization without homodimers.*

### 17.27.1 Detailed Description

Equilibrium Probability implementations.

This file includes various implementations for equilibrium probability computations based on the partition function of an RNA sequence, two concatenated sequences, or a sequence alignment.

## 17.28 ViennaRNA/eval.h File Reference

Functions and variables related to energy evaluation of sequence/structure pairs.

Include dependency graph for eval.h:

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

## Functions

### Basic Energy Evaluation Interface with Dot-Bracket Structure String

- float [vrna_eval_structure](#) ([vrna_fold_compound_t](#) ∗vc, const char ∗structure)

  *Calculate the free energy of an already folded RNA.*
- float [vrna_eval_covar_structure](#) ([vrna_fold_compound_t](#) ∗vc, const char ∗structure)

  *Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.*
- float [vrna_eval_structure_verbose](#) ([vrna_fold_compound_t](#) ∗vc, const char ∗structure, FILE ∗file)

  *Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*
- float [vrna_eval_structure_v](#) ([vrna_fold_compound_t](#) ∗vc, const char ∗structure, int verbosity_level, FILE ∗file)

  *Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*

### Basic Energy Evaluation Interface with Structure Pair Table

- int [vrna_eval_structure_pt](#) ([vrna_fold_compound_t](#) ∗vc, const short ∗pt)

  *Calculate the free energy of an already folded RNA.*
- int [vrna_eval_structure_pt_verbose](#) ([vrna_fold_compound_t](#) ∗vc, const short ∗pt, FILE ∗file)

  *Calculate the free energy of an already folded RNA.*
- int [vrna_eval_structure_pt_v](#) ([vrna_fold_compound_t](#) ∗vc, const short ∗pt, int verbosity_level, FILE ∗file)

  *Calculate the free energy of an already folded RNA.*

### Simplified Energy Evaluation with Sequence and Dot-Bracket Strings

- float [vrna_eval_structure_simple](#) (const char ∗string, const char ∗structure)

  *Calculate the free energy of an already folded RNA.*
- float [vrna_eval_circ_structure](#) (const char ∗string, const char ∗structure)

  *Evaluate the free energy of a sequence/structure pair where the sequence is circular.*
- float [vrna_eval_gquad_structure](#) (const char ∗string, const char ∗structure)

  *Evaluate the free energy of a sequence/structure pair where the structure may contain G-Quadruplexes.*
- float [vrna_eval_circ_gquad_structure](#) (const char ∗string, const char ∗structure)

  *Evaluate the free energy of a sequence/structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*
- float [vrna_eval_structure_simple_verbose](#) (const char ∗string, const char ∗structure, FILE ∗file)

  *Calculate the free energy of an already folded RNA and print contributions per loop.*
- float [vrna_eval_structure_simple_v](#) (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

  *Calculate the free energy of an already folded RNA and print contributions per loop.*
- float [vrna_eval_circ_structure_v](#) (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

  *Evaluate free energy of a sequence/structure pair, assume sequence to be circular and print contributions per loop.*
- float [vrna_eval_gquad_structure_v](#) (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

  *Evaluate free energy of a sequence/structure pair, allow for G-Quadruplexes in the structure and print contributions per loop.*
- float [vrna_eval_circ_gquad_structure_v](#) (const char ∗string, const char ∗structure, int verbosity_level, FILE ∗file)

  *Evaluate free energy of a sequence/structure pair, assume sequence to be circular, allow for G-Quadruplexes in the structure, and print contributions per loop.*

### Simplified Energy Evaluation with Sequence Alignments and Consensus Structure Dot-Bracket String

- float [vrna_eval_consensus_structure_simple](#) (const char ∗∗alignment, const char ∗structure)

  *Calculate the free energy of an already folded RNA sequence alignment.*
- float [vrna_eval_circ_consensus_structure](#) (const char ∗∗alignment, const char ∗structure)

  *Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequences are circular.*

- float vrna_eval_gquad_consensus_structure (const char ∗∗alignment, const char ∗structure)

    *Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the structure may contain G-Quadruplexes.*

- float vrna_eval_circ_gquad_consensus_structure (const char ∗∗alignment, const char ∗structure)

    *Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*

- float vrna_eval_consensus_structure_simple_verbose (const char ∗∗alignment, const char ∗structure, F↩ILE ∗file)

    *Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*

- float vrna_eval_consensus_structure_simple_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*

- float vrna_eval_circ_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences and print contributions per loop.*

- float vrna_eval_gquad_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate the free energy of a consensus structure for an RNA sequence alignment, allow for annotated G-↩Quadruplexes in the structure and print contributions per loop.*

- float vrna_eval_circ_gquad_consensus_structure_v (const char ∗∗alignment, const char ∗structure, int verbosity_level, FILE ∗file)

    *Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences, allow for annotated G-Quadruplexes in the structure and print contributions per loop.*

### Simplified Energy Evaluation with Sequence String and Structure Pair Table

- int vrna_eval_structure_pt_simple (const char ∗string, const short ∗pt)

    *Calculate the free energy of an already folded RNA.*

- int vrna_eval_structure_pt_simple_verbose (const char ∗string, const short ∗pt, FILE ∗file)

    *Calculate the free energy of an already folded RNA.*

- int vrna_eval_structure_pt_simple_v (const char ∗string, const short ∗pt, int verbosity_level, FILE ∗file)

    *Calculate the free energy of an already folded RNA.*

### Simplified Energy Evaluation with Sequence Alignment and Consensus Structure Pair Table

- int vrna_eval_consensus_structure_pt_simple (const char ∗∗alignment, const short ∗pt)

    *Evaluate the Free Energy of a Consensus Secondary Structure given a Sequence Alignment.*

- int **vrna_eval_consensus_structure_pt_simple_verbose** (const char ∗∗alignment, const short ∗pt, FILE ∗file)

- int **vrna_eval_consensus_structure_pt_simple_v** (const char ∗∗alignment, const short ∗pt, int verbosity_level, FILE ∗file)

### Basic Loop Energy Evaluation with Structure Pair Table

- int vrna_eval_loop_pt (vrna_fold_compound_t ∗vc, int i, const short ∗pt)

    *Calculate energy of a loop.*

- int vrna_eval_loop_pt_v (vrna_fold_compound_t ∗vc, int i, const short ∗pt, int verbosity_level)

    *Calculate energy of a loop.*

### Basic Energy Evaluation for atomic moves

*Here, atomic moves are considered opening or closing a single base pair, as well as shifting one pairing partner of an existing pair to a different location*

- float vrna_eval_move (vrna_fold_compound_t ∗vc, const char ∗structure, int m1, int m2)

*Calculate energy of a move (closing or opening of a base pair)*
- int vrna_eval_move_pt (vrna_fold_compound_t *vc, short *pt, int m1, int m2)

    *Calculate energy of a move (closing or opening of a base pair)*
- int **vrna_eval_move_pt_simple** (const char *string, short *pt, int m1, int m2)
- int **vrna_eval_move_shift_pt** (vrna_fold_compound_t *vc, vrna_move_t *m, short *structure)

**Deprecated Energy Evaluation functions**

*Using the functions below is discouraged as they have been marked deprecated and will be removed from the library in the (near) future!*

- 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, vrna_param_t *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, vrna_param_t *parameters, int verbosity_level)

    *Calculate the free energy of an already folded circular RNA.*
- float **energy_of_gquad_structure** (const char *string, const char *structure, int verbosity_level)
- float **energy_of_gquad_struct_par** (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)
- 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, vrna_param_t *parameters, int verbosity_level)

    *Calculate the free energy of an already folded RNA.*
- 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.*
- 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 cut_point

    *set to first pos of second seq for cofolding*
- int eos_debug

    *verbose info from energy_of_struct*

### 17.28.1  Detailed Description

Functions and variables related to energy evaluation of sequence/structure pairs.

## 17.29 ViennaRNA/exterior_loops.h File Reference

Energy evaluation of exterior loops for MFE and partition function calculations.

Include dependency graph for exterior_loops.h:

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

### Data Structures

- struct vrna_mx_pf_aux_el_t

    *Auxiliary helper arrays for fast exterior loop computations. More...*

### Functions

- int vrna_E_ext_stem (unsigned int type, int n5d, int n3d, vrna_param_t *p)

    *Evaluate a stem branching off the exterior loop.*
- FLT_OR_DBL vrna_exp_E_ext_stem (unsigned int type, int n5d, int n3d, vrna_exp_param_t *p)

    *Evaluate a stem branching off the exterior loop (Boltzmann factor version)*
- int E_Stem (int type, int si1, int sj1, int extLoop, vrna_param_t *P)

    *Compute the energy contribution of a stem branching off a loop-region.*
- FLT_OR_DBL exp_E_ExtLoop (int type, int si1, int sj1, vrna_exp_param_t *P)
- FLT_OR_DBL exp_E_Stem (int type, int si1, int sj1, int extLoop, vrna_exp_param_t *P)

### 17.29.1 Detailed Description

Energy evaluation of exterior loops for MFE and partition function calculations.

## 17.30 ViennaRNA/file_formats.h File Reference

Read and write different file formats for RNA sequences, structures.

Include dependency graph for file_formats.h:

### Macros

- #define VRNA_OPTION_MULTILINE 32U

    *Tell a function that an input is assumed to span several lines.*
- #define VRNA_CONSTRAINT_MULTILINE 32U

    *parse multiline constraint*

**Functions**

- void vrna_file_helixlist (const char *seq, const char *db, float energy, FILE *file)

  *Print a secondary structure as helix list.*

- void vrna_file_connect (const char *seq, const char *db, float energy, const char *identifier, FILE *file)

  *Print a secondary structure as connect table.*

- void vrna_file_bpseq (const char *seq, const char *db, FILE *file)

  *Print a secondary structure in bpseq format.*

- void vrna_file_json (const char *seq, const char *db, double energy, const char *identifier, FILE *file)

  *Print a secondary structure in jsonformat.*

- unsigned int vrna_file_fasta_read_record (char **header, char **sequence, char ***rest, FILE *file, unsigned int options)

  *Get a (fasta) data set from a file or stdin.*

- char * vrna_extract_record_rest_structure (const char **lines, unsigned int length, unsigned int option)

  *Extract a dot-bracket structure string from (multiline)character array.*

- int vrna_file_SHAPE_read (const char *file_name, int length, double default_value, char *sequence, double *values)

  *Read data from a given SHAPE reactivity input file.*

- void vrna_extract_record_rest_constraint (char **cstruc, const char **lines, unsigned int option)

  *Extract a hard constraint encoded as pseudo dot-bracket string.*

- unsigned int read_record (char **header, char **sequence, char ***rest, unsigned int options)

  *Get a data record from stdin.*

### 17.30.1 Detailed Description

Read and write different file formats for RNA sequences, structures.

## 17.31 ViennaRNA/file_formats_msa.h File Reference

Functions dealing with file formats for Multiple Sequence Alignments (MSA)

Include dependency graph for file_formats_msa.h:

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

**Macros**

- #define VRNA_FILE_FORMAT_MSA_CLUSTAL 1U

  *Option flag indicating ClustalW formatted files.*
- #define VRNA_FILE_FORMAT_MSA_STOCKHOLM 2U

  *Option flag indicating Stockholm 1.0 formatted files.*
- #define VRNA_FILE_FORMAT_MSA_FASTA 4U

  *Option flag indicating FASTA (Pearson) formatted files.*
- #define VRNA_FILE_FORMAT_MSA_MAF 8U

  *Option flag indicating MAF formatted files.*
- #define VRNA_FILE_FORMAT_MSA_MIS 16U

  *Option flag indicating most informative sequence (MIS) output.*
- #define VRNA_FILE_FORMAT_MSA_DEFAULT

  *Option flag indicating the set of default file formats.*
- #define VRNA_FILE_FORMAT_MSA_NOCHECK 4096U

  *Option flag to disable validation of the alignment.*
- #define VRNA_FILE_FORMAT_MSA_UNKNOWN 8192U

  *Return flag of vrna_file_msa_detect_format() to indicate unknown or malformatted alignment.*
- #define VRNA_FILE_FORMAT_MSA_APPEND 16384U

  *Option flag indicating to append data to a multiple sequence alignment file rather than overwriting it.*
- #define VRNA_FILE_FORMAT_MSA_QUIET 32768U

  *Option flag to suppress unnecessary spam messages on* `stderr`
- #define VRNA_FILE_FORMAT_MSA_SILENT 65536U

  *Option flag to completely silence any warnings on* `stderr`

**Functions**

- int vrna_file_msa_read (const char ∗filename, char ∗∗∗names, char ∗∗∗aln, char ∗∗id, char ∗∗structure, unsigned int options)

  *Read a multiple sequence alignment from file.*
- int vrna_file_msa_read_record (FILE ∗fp, char ∗∗∗names, char ∗∗∗aln, char ∗∗id, char ∗∗structure, unsigned int options)

  *Read a multiple sequence alignment from file handle.*
- unsigned int vrna_file_msa_detect_format (const char ∗filename, unsigned int options)

  *Detect the format of a multiple sequence alignment file.*
- int vrna_file_msa_write (const char ∗filename, const char ∗∗names, const char ∗∗aln, const char ∗id, const char ∗structure, const char ∗source, unsigned int options)

  *Write multiple sequence alignment file.*

### 17.31.1 Detailed Description

Functions dealing with file formats for Multiple Sequence Alignments (MSA)

## 17.32 ViennaRNA/file_utils.h File Reference

Several utilities for file handling.

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

**Functions**

- void vrna_file_copy (FILE ∗from, FILE ∗to)

  *Inefficient 'cp'.*
- char ∗ vrna_read_line (FILE ∗fp)

  *Read a line of arbitrary length from a stream.*
- int vrna_mkdir_p (const char ∗path)

  *Recursivly create a directory tree.*
- char ∗ vrna_basename (const char ∗path)

  *Extract the filename from a file path.*
- char ∗ vrna_dirname (const char ∗path)

  *Extract the directory part of a file path.*
- char ∗ vrna_filename_sanitize (const char ∗name, const char ∗replacement)

  *Sanitize a file name.*

### 17.32.1 Detailed Description

Several utilities for file handling.

## 17.33 ViennaRNA/findpath.h File Reference

A breadth-first search heuristic for optimal direct folding paths.

Include dependency graph for findpath.h:

**Data Structures**

- struct vrna_path_s

  *An element of a refolding path list. More...*

**Typedefs**

- typedef struct vrna_path_s vrna_path_t

  *Typename for the refolding path data structure vrna_path_s.*
- typedef struct vrna_path_s path_t

  *Old typename of vrna_path_s.*

**Functions**

- int vrna_path_findpath_saddle (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width)

    *Find energy of a saddle point between 2 structures (search only direct path)*

- int vrna_path_findpath_saddle_ub (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width, int maxE)

    *Find energy of a saddle point between 2 structures (search only direct path)*

- vrna_path_t ∗ vrna_path_findpath (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width)

    *Find refolding path between 2 structures (search only direct path)*

- vrna_path_t ∗ vrna_path_findpath_ub (vrna_fold_compound_t ∗vc, const char ∗s1, const char ∗s2, int width, int maxE)

    *Find refolding path between 2 structures (search only direct path)*

- int find_saddle (const char ∗seq, const char ∗s1, const char ∗s2, int width)

    *Find energy of a saddle point between 2 structures (search only direct path)*

- void free_path (vrna_path_t ∗path)

    *Free memory allocated by get_path() function.*

- vrna_path_t ∗ get_path (const char ∗seq, const char ∗s1, const char ∗s2, int width)

    *Find refolding path between 2 structures (search only direct path)*

### 17.33.1 Detailed Description

A breadth-first search heuristic for optimal direct folding paths.

## 17.34 ViennaRNA/fold.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for fold.h:

**Functions**

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

    *Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.*

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

    *Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.*

- float fold_par (const char ∗sequence, char ∗structure, vrna_param_t ∗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.*

- void update_fold_params_par (vrna_param_t ∗parameters)

    *Recalculate energy parameters.*
- void export_fold_arrays (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p)
- void export_fold_arrays_par (int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p, vrna_param_t ∗∗P_p)
- void export_circfold_arrays (int ∗Fc_p, int ∗FcH_p, int ∗FcI_p, int ∗FcM_p, int ∗∗fM2_p, int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p)
- void export_circfold_arrays_par (int ∗Fc_p, int ∗FcH_p, int ∗FcI_p, int ∗FcM_p, int ∗∗fM2_p, int ∗∗f5_p, int ∗∗c_p, int ∗∗fML_p, int ∗∗fM1_p, int ∗∗indx_p, char ∗∗ptype_p, vrna_param_t ∗∗P_p)
- 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)

### 17.34.1 Detailed Description

MFE calculations for single RNA sequences.

## 17.35 ViennaRNA/fold_compound.h File Reference

The Basic Fold Compound API.

Include dependency graph for fold_compound.h:

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

### Data Structures

- struct vrna_fc_s

    *The most basic data structure required by many functions throughout the RNAlib. More...*

### Macros

- #define VRNA_STATUS_MFE_PRE (unsigned char)1

    *Status message indicating that MFE computations are about to begin.*
- #define VRNA_STATUS_MFE_POST (unsigned char)2

    *Status message indicating that MFE computations are finished.*
- #define VRNA_STATUS_PF_PRE (unsigned char)3

    *Status message indicating that Partition function computations are about to begin.*
- #define VRNA_STATUS_PF_POST (unsigned char)4

    *Status message indicating that Partition function computations are finished.*
- #define VRNA_OPTION_DEFAULT 0U

    *Option flag to specify default settings/requirements.*
- #define VRNA_OPTION_MFE 1U

*Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.*

- #define VRNA_OPTION_PF 2U

    *Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.*

- #define VRNA_OPTION_HYBRID 4U

    *Option flag to specify requirement of dimer DP matrices.*

- #define VRNA_OPTION_EVAL_ONLY 8U

    *Option flag to specify that neither MFE, nor PF DP matrices are required.*

- #define VRNA_OPTION_WINDOW 16U

    *Option flag to specify requirement of DP matrices for local folding approaches.*

## Typedefs

- typedef struct vrna_fc_s vrna_fold_compound_t

    *Typename for the fold_compound data structure vrna_fc_s.*

- typedef void() vrna_callback_free_auxdata(void ∗data)

    *Callback to free memory allocated for auxiliary user-provided data.*

- typedef void() vrna_callback_recursion_status(unsigned char status, void ∗data)

    *Callback to perform specific user-defined actions before, or after recursive computations.*

## Enumerations

- enum vrna_fc_type_e { VRNA_FC_TYPE_SINGLE, VRNA_FC_TYPE_COMPARATIVE }

    *An enumerator that is used to specify the type of a vrna_fold_compound_t.*

## Functions

- vrna_fold_compound_t ∗ vrna_fold_compound (const char ∗sequence, vrna_md_t ∗md_p, unsigned int options)

    *Retrieve a vrna_fold_compound_t data structure for single sequences and hybridizing sequences.*

- vrna_fold_compound_t ∗ vrna_fold_compound_comparative (const char ∗∗sequences, vrna_md_t ∗md_p, unsigned int options)

    *Retrieve a vrna_fold_compound_t data structure for sequence alignments.*

- void vrna_fold_compound_free (vrna_fold_compound_t ∗vc)

    *Free memory occupied by a vrna_fold_compound_t.*

- void vrna_fold_compound_add_auxdata (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗f)

    *Add auxiliary data to the vrna_fold_compound_t.*

- void vrna_fold_compound_add_callback (vrna_fold_compound_t ∗vc, vrna_callback_recursion_status ∗f)

    *Add a recursion status callback to the vrna_fold_compound_t.*

### 17.35.1  Detailed Description

The Basic Fold Compound API.

## 17.36 ViennaRNA/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:

**Variables**

- int fold_constrained

  *Global switch to activate/deactivate folding with structure constraints.*
- int csv

  *generate comma seperated output*
- char ∗ RibosumFile
- 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().*
- FLT_OR_DBL ∗ pr

  *A pointer to the base pair probability matrix.*
- int ∗ iindx

  *index array to move through pr.*

### 17.36.1 Detailed Description

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

### 17.36.2 Variable Documentation

#### 17.36.2.1 RibosumFile

```
char* RibosumFile
```

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

**17.36.2.2 james_rule**

```
int james_rule
```

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

**17.36.2.3 logML**

```
int logML
```

use logarithmic multiloop energy function

**17.36.2.4 cut_point**

```
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 vrna_file_PS_rnaplot() and PS_dot_plot() to mark the chain break in postscript plots.

**17.36.2.5 base_pair**

```
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!

**17.36.2.6 pr**

```
FLT_OR_DBL* pr
```

A pointer to the base pair probability matrix.

**Deprecated** Do not use this variable anymore!

**17.36.2.7 iindx**

```
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!

## 17.37 ViennaRNA/gquad.h File Reference

G-quadruplexes.

Include dependency graph for gquad.h:

**Functions**

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

### 17.37.1 Detailed Description

G-quadruplexes.

## 17.38 ViennaRNA/grammar.h File Reference

Implementations for the RNA folding grammar.

Include dependency graph for grammar.h:

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

**Data Structures**

- struct vrna_gr_aux_s

**17.38.1 Detailed Description**

Implementations for the RNA folding grammar.

## 17.39 ViennaRNA/hairpin_loops.h File Reference

Energy evaluation of hairpin loops for MFE and partition function calculations.

Include dependency graph for hairpin_loops.h:

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

**Functions**

- PRIVATE int E_Hairpin (int size, int type, int si1, int sj1, const char ∗string, vrna_param_t ∗P)

  *Compute the Energy of a hairpin-loop.*

- PRIVATE FLT_OR_DBL exp_E_Hairpin (int u, int type, short si1, short sj1, const char ∗string, vrna_exp_param_t ∗P)

  *Compute Boltzmann weight $e^{-\Delta G/kT}$ of a hairpin loop.*

- int vrna_E_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.*

- int vrna_E_ext_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.*

- int vrna_eval_ext_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate free energy of an exterior hairpin loop.*

- int vrna_eval_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *Evaluate free energy of a hairpin loop.*

- FLT_OR_DBL vrna_exp_E_hp_loop (vrna_fold_compound_t ∗vc, int i, int j)

  *High-Level function for hairpin loop energy evaluation (partition function variant)*

- int vrna_BT_hp_loop (vrna_fold_compound_t ∗vc, int i, int j, int en, vrna_bp_stack_t ∗bp_stack, int ∗stack← _count)

  *Backtrack a hairpin loop closed by $(i, j)$.*

**17.39.1 Detailed Description**

Energy evaluation of hairpin loops for MFE and partition function calculations.

## 17.40 ViennaRNA/interior_loops.h File Reference

Energy evaluation of interior loops for MFE and partition function calculations.

Include dependency graph for interior_loops.h:

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

### Functions

- PRIVATE int E_IntLoop (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t ∗P)
- PRIVATE FLT_OR_DBL exp_E_IntLoop (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, vrna_exp_param_t ∗P)
- int vrna_eval_int_loop (vrna_fold_compound_t ∗vc, int i, int j, int k, int l)
    *Evaluate the free energy contribution of an interior loop with delimiting base pairs* $(i, j)$ *and* $(k, l)$.
- int vrna_BT_stack (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int ∗en, vrna_bp_stack_t ∗bp_stack, int ∗stack↩_count)
    *Backtrack a stacked pair closed by* $(i, j)$.
- int vrna_BT_int_loop (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int en, vrna_bp_stack_t ∗bp_stack, int ∗stack_count)
    *Backtrack an interior loop closed by* $(i, j)$.

### 17.40.1 Detailed Description

Energy evaluation of interior loops for MFE and partition function calculations.

## 17.41 ViennaRNA/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

### 17.41.1 Detailed Description

Inverse folding routines.

## 17.42 ViennaRNA/Lfold.h File Reference

Functions for locally optimal MFE structure prediction.

Include dependency graph for Lfold.h:

**Functions**

- float vrna_Lfold (const char ∗string, int window_size, FILE ∗file)

  *Local MFE prediction using a sliding window approach (simplified interface)*
- float vrna_Lfoldz (const char ∗string, int window_size, double min_z, FILE ∗file)

  *Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)*
- 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 ∗∗AS, char ∗structure, int maxdist)

### 17.42.1 Detailed Description

Functions for locally optimal MFE structure prediction.

## 17.43 ViennaRNA/loop_energies.h File Reference

Energy evaluation for MFE and partition function calculations.

Include dependency graph for loop_energies.h:

### 17.43.1 Detailed Description

Energy evaluation for MFE and partition function calculations.

This file contains functions for the calculation of the free energy $\Delta G$ 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() ].

## 17.44 ViennaRNA/LPfold.h File Reference

Partition function implementation for the Lfold algorithm.

Include dependency graph for LPfold.h:

### Macros

- #define VRNA_PROBS_WINDOW_BPP 4096U

    *Option flag to activate callback triggers for pairing probabilities.*
- #define VRNA_PROBS_WINDOW_UP 8192U

    *Option flag to activate callback triggers for unpaired probabilities.*
- #define VRNA_PROBS_WINDOW_STACKP 16384U

    *Option flag to activate callback triggers for stacking probabilities.*
- #define VRNA_PROBS_WINDOW_UP_SPLIT 32768U

    *Option flag to activate callback triggers for unpaired probabilities (split up into different loop types)*

### Typedefs

- typedef void() vrna_probs_window_callback(FLT_OR_DBL ∗pr, int pr_size, int i, int max, unsigned int type, void ∗data)

    *Sliding window probability computation callback.*

### Functions

- vrna_ep_t ∗ vrna_pfl_fold (const char ∗sequence, int window_size, int max_bp_span, float cutoff)

    *Compute base pair probabilities using a sliding-window approach.*
- void vrna_pfl_fold_cb (const char ∗sequence, int window_size, int max_bp_span, vrna_probs_window_callback ∗cb, void ∗data)

    *Compute base pair probabilities using a sliding-window approach (callback version)*
- double ∗∗ vrna_pfl_fold_up (const char ∗sequence, int ulength, int window_size, int max_bp_span)

    *Compute probability of contiguous unpaired segments.*
- void vrna_pfl_fold_up_cb (const char ∗sequence, int ulength, int window_size, int max_bp_span, vrna_probs_window_callback ∗cb, void ∗data)

    *Compute probability of contiguous unpaired segments.*
- void vrna_probs_window (vrna_fold_compound_t ∗fc, int ulength, unsigned int options, vrna_probs_window_callback ∗cb, void ∗data)

    *Compute various equilibrium probabilities under a sliding window approach.*
- void update_pf_paramsLP (int length)
- vrna_ep_t ∗ pfl_fold (char ∗sequence, int winSize, int pairSize, float cutoffb, double ∗∗pU, vrna_ep_t ∗∗dpp2, FILE ∗pUfp, FILE ∗spup)

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

## 17.44.1 Detailed Description

Partition function implementation for the Lfold algorithm.

## 17.44.2 Typedef Documentation

### 17.44.2.1 vrna_probs_window_callback

```
typedef void() vrna_probs_window_callback(FLT_OR_DBL *pr, int pr_size, int i, int max, unsigned
int type, void *data)
```

Sliding window probability computation callback.

**Notes on Callback Functions** This function will be called for each probability data set in the sliding window probability computation implementation of vrna_probs_window(). The argument *type* specifies the type of probability that is passed to this function.

**See also**

vrna_probs_window(), vrna_pfl_fold_up_cb() VRNA_PROBS_WINDOW_BPP, VRNA_PROBS_WINDOW_UP, VRNA_PROBS_WINDOW_STACKP, #VRNA_EXT_LOOP, #VRNA_HP_LOOP, #VRNA_INT_LOOP, #VR↩ NA_MB_LOOP, #VRNA_ANY_LOOP

**Parameters**

| pr | An array of probabilities |
|---|---|
| pr_size | The length of the probability array |
| i | The i-position (5') of the probabilities |
| max | The (theoretical) maximum length of the probability array |
| data | Auxiliary data |

## 17.44.3 Function Documentation

### 17.44.3.1 init_pf_foldLP()

```
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

## 17.45 ViennaRNA/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.*

### 17.45.1 Detailed Description

Computes a MEA (maximum expected accuracy) structure.

## 17.46 ViennaRNA/mfe.h File Reference

Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.

Include dependency graph for mfe.h:

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

### Typedefs

- typedef void() vrna_mfe_window_callback(int start, int end, const char ∗structure, float en, void ∗data)

  *The default callback for sliding window MFE structure predictions.*

### Functions

- float vrna_mfe (vrna_fold_compound_t ∗vc, char ∗structure)

  *Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.*

- float vrna_mfe_dimer (vrna_fold_compound_t ∗vc, char ∗structure)

  *Compute the minimum free energy of two interacting RNA molecules.*

- float vrna_mfe_window (vrna_fold_compound_t ∗vc, FILE ∗file)

  *Local MFE prediction using a sliding window approach.*

- float vrna_mfe_window_zscore (vrna_fold_compound_t ∗vc, double min_z, FILE ∗file)

  *Local MFE prediction using a sliding window approach (with z-score cut-off)*

### 17.46.1  Detailed Description

Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.

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

## 17.47  ViennaRNA/mm.h File Reference

Several Maximum Matching implementations.

### 17.47.1  Detailed Description

Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

## 17.48  ViennaRNA/model.h File Reference

The model details data structure and its corresponding modifiers.

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

**Data Structures**

- struct vrna_md_s

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

**Macros**

- #define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0

    *Default temperature for structure prediction and free energy evaluation in $^{\circ}C$*
- #define VRNA_MODEL_DEFAULT_PF_SCALE -1

    *Default scaling factor for partition function computations.*
- #define VRNA_MODEL_DEFAULT_BETA_SCALE 1.

    *Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.*
- #define VRNA_MODEL_DEFAULT_DANGLES 2

    *Default dangling end model.*
- #define VRNA_MODEL_DEFAULT_SPECIAL_HP 1

    *Default model behavior for lookup of special tri-, tetra-, and hexa-loops.*
- #define VRNA_MODEL_DEFAULT_NO_LP 0

    *Default model behavior for so-called 'lonely pairs'.*
- #define VRNA_MODEL_DEFAULT_NO_GU 0

    *Default model behavior for G-U base pairs.*

- #define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0

    *Default model behavior for G-U base pairs closing a loop.*
- #define VRNA_MODEL_DEFAULT_CIRC 0

    *Default model behavior to treat a molecule as a circular RNA (DNA)*
- #define VRNA_MODEL_DEFAULT_GQUAD 0

    *Default model behavior regarding the treatment of G-Quadruplexes.*
- #define VRNA_MODEL_DEFAULT_UNIQ_ML 0

    *Default behavior of the model regarding unique multi-branch loop decomposition.*
- #define VRNA_MODEL_DEFAULT_ENERGY_SET 0

    *Default model behavior on which energy set to use.*
- #define VRNA_MODEL_DEFAULT_BACKTRACK 1

    *Default model behavior with regards to backtracking of structures.*
- #define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'

    *Default model behavior on what type of backtracking to perform.*
- #define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1

    *Default model behavior with regards to computing base pair probabilities.*
- #define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1

    *Default model behavior for the allowed maximum base pair span.*
- #define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1

    *Default model behavior for the sliding window approach.*
- #define VRNA_MODEL_DEFAULT_LOG_ML 0

    *Default model behavior on how to evaluate the energy contribution of multi-branch loops.*
- #define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0

    *Default model behavior for consensus structure energy evaluation.*
- #define VRNA_MODEL_DEFAULT_ALI_RIBO 0

    *Default model behavior for consensus structure co-variance contribution assessment.*
- #define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.

    *Default model behavior for weighting the co-variance score in consensus structure prediction.*
- #define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.

    *Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.*
- #define MAXALPHA 20

    *Maximal length of alphabet.*

## Typedefs

- typedef struct vrna_md_s vrna_md_t

    *Typename for the model details data structure vrna_md_s.*

## Functions

- void vrna_md_set_default (vrna_md_t ∗md)

    *Apply default model details to a provided vrna_md_t data structure.*
- void vrna_md_update (vrna_md_t ∗md)

    *Update the model details data structure.*
- vrna_md_t ∗ vrna_md_copy (vrna_md_t ∗md_to, const vrna_md_t ∗md_from)

    *Copy/Clone a vrna_md_t model.*
- char ∗ vrna_md_option_string (vrna_md_t ∗md)

    *Get a corresponding commandline parameter string of the options in a vrna_md_t.*
- void vrna_md_defaults_reset (vrna_md_t ∗md_p)

*Reset the global default model details to a specific set of parameters, or their initial values.*

- void vrna_md_defaults_temperature (double T)

    *Set default temperature for energy evaluation of loops.*

- double vrna_md_defaults_temperature_get (void)

    *Get default temperature for energy evaluation of loops.*

- void vrna_md_defaults_betaScale (double b)

    *Set default scaling factor of thermodynamic temperature in Boltzmann factors.*

- double vrna_md_defaults_betaScale_get (void)

    *Get default scaling factor of thermodynamic temperature in Boltzmann factors.*

- void vrna_md_defaults_dangles (int d)

    *Set default dangle model for structure prediction.*

- int vrna_md_defaults_dangles_get (void)

    *Get default dangle model for structure prediction.*

- void vrna_md_defaults_special_hp (int flag)

    *Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*

- int vrna_md_defaults_special_hp_get (void)

    *Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*

- void vrna_md_defaults_noLP (int flag)

    *Set default behavior for prediction of canonical secondary structures.*

- int vrna_md_defaults_noLP_get (void)

    *Get default behavior for prediction of canonical secondary structures.*

- void vrna_md_defaults_noGU (int flag)

    *Set default behavior for treatment of G-U wobble pairs.*

- int vrna_md_defaults_noGU_get (void)

    *Get default behavior for treatment of G-U wobble pairs.*

- void vrna_md_defaults_noGUclosure (int flag)

    *Set default behavior for G-U pairs as closing pair for loops.*

- int vrna_md_defaults_noGUclosure_get (void)

    *Get default behavior for G-U pairs as closing pair for loops.*

- void vrna_md_defaults_logML (int flag)

    *Set default behavior recomputing free energies of multi-branch loops using a logarithmic model.*

- int vrna_md_defaults_logML_get (void)

    *Get default behavior recomputing free energies of multi-branch loops using a logarithmic model.*

- void vrna_md_defaults_circ (int flag)

    *Set default behavior whether input sequences are circularized.*

- int vrna_md_defaults_circ_get (void)

    *Get default behavior whether input sequences are circularized.*

- void vrna_md_defaults_gquad (int flag)

    *Set default behavior for treatment of G-Quadruplexes.*

- int vrna_md_defaults_gquad_get (void)

    *Get default behavior for treatment of G-Quadruplexes.*

- void vrna_md_defaults_uniq_ML (int flag)

    *Set default behavior for creating additional matrix for unique multi-branch loop prediction.*

- int vrna_md_defaults_uniq_ML_get (void)

    *Get default behavior for creating additional matrix for unique multi-branch loop prediction.*

- void vrna_md_defaults_energy_set (int e)

    *Set default energy set.*

- int vrna_md_defaults_energy_set_get (void)

    *Get default energy set.*

- void vrna_md_defaults_backtrack (int flag)

    *Set default behavior for whether to backtrack secondary structures.*

- int vrna_md_defaults_backtrack_get (void)

    *Get default behavior for whether to backtrack secondary structures.*
- void vrna_md_defaults_backtrack_type (char t)

    *Set default backtrack type, i.e. which DP matrix is used.*
- char vrna_md_defaults_backtrack_type_get (void)

    *Get default backtrack type, i.e. which DP matrix is used.*
- void vrna_md_defaults_compute_bpp (int flag)

    *Set the default behavior for whether to compute base pair probabilities after partition function computation.*
- int vrna_md_defaults_compute_bpp_get (void)

    *Get the default behavior for whether to compute base pair probabilities after partition function computation.*
- void vrna_md_defaults_max_bp_span (int span)

    *Set default maximal base pair span.*
- int vrna_md_defaults_max_bp_span_get (void)

    *Get default maximal base pair span.*
- void vrna_md_defaults_min_loop_size (int size)

    *Set default minimal loop size.*
- int vrna_md_defaults_min_loop_size_get (void)

    *Get default minimal loop size.*
- void vrna_md_defaults_window_size (int size)

    *Set default window size for sliding window structure prediction approaches.*
- int vrna_md_defaults_window_size_get (void)

    *Get default window size for sliding window structure prediction approaches.*
- void vrna_md_defaults_oldAliEn (int flag)

    *Set default behavior for whether to use old energy model for comparative structure prediction.*
- int vrna_md_defaults_oldAliEn_get (void)

    *Get default behavior for whether to use old energy model for comparative structure prediction.*
- void vrna_md_defaults_ribo (int flag)

    *Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.*
- int vrna_md_defaults_ribo_get (void)

    *Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.*
- void vrna_md_defaults_cv_fact (double factor)

    *Set the default co-variance scaling factor used in comparative structure prediction.*
- double vrna_md_defaults_cv_fact_get (void)

    *Get the default co-variance scaling factor used in comparative structure prediction.*
- void vrna_md_defaults_nc_fact (double factor)
- double vrna_md_defaults_nc_fact_get (void)
- void vrna_md_defaults_sfact (double factor)

    *Set the default scaling factor used to avoid under-/overflows in partition function computation.*
- double vrna_md_defaults_sfact_get (void)

    *Get the default scaling factor used to avoid under-/overflows in partition function computation.*
- void set_model_details (vrna_md_t ∗md)

    *Set default model details.*

## Variables

- double temperature

    *Rescale energy parameters to a temperature in degC.*
- double pf_scale

    *A scaling factor used by pf_fold() to avoid overflows.*
- int dangles

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

- int tetra_loop

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

- int noLonelyPairs

    *Global switch to avoid/allow helices of length 1.*

- 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 circ

    *backward compatibility variable.. this does not effect anything*

- int gquad

    *Allow G-quadruplex formation.*

- int uniq_ML

    *do ML decomposition uniquely (for subopt)*

- int energy_set

    *0 = BP; 1=any with GC; 2=any with AU-parameter*

- int do_backtrack

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

- char backtrack_type

    *A backtrack array marker for inverse_fold()*

- char ∗ nonstandards

    *contains allowed non standard base pairs*

- int max_bp_span

    *Maximum allowed base pair span.*

- int oldAliEn

    *use old alifold energies (with gaps)*

- int ribo

    *use ribosum matrices*

- int logML

    *if nonzero use logarithmic ML energy in energy_of_struct*

## 17.48.1 Detailed Description

The model details data structure and its corresponding modifiers.

## 17.49 ViennaRNA/multibranch_loops.h File Reference

Energy evaluation of multibranch loops for MFE and partition function calculations.

Include dependency graph for multibranch_loops.h:

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

**Data Structures**

- struct vrna_mx_pf_aux_ml_t

    *Auxiliary helper arrays for fast exterior loop computations. More...*

**Functions**

- int vrna_E_mb_loop_stack (vrna_fold_compound_t ∗vc, int i, int j)

    *Evaluate energy of a multi branch helices stacking onto closing pair (i,j)*

- int vrna_BT_mb_loop (vrna_fold_compound_t ∗vc, int ∗i, int ∗j, int ∗k, int en, int ∗component1, int ∗component2)

    *Backtrack the decomposition of a multi branch loop closed by* $(i, j)$*.*

### 17.49.1 Detailed Description

Energy evaluation of multibranch loops for MFE and partition function calculations.

## 17.50 ViennaRNA/naview.h File Reference

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

## 17.51 ViennaRNA/neighbor.h File Reference

Methods to compute the neighbors of an RNA secondary structure.

Include dependency graph for neighbor.h:

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

**Data Structures**

- struct vrna_move_s

    *An atomic representation of the transition / move from one structure to its neighbor. More...*

**Macros**

- #define VRNA_MOVESET_INSERTION 4

    *Option flag indicating insertion move.*
- #define VRNA_MOVESET_DELETION 8

    *Option flag indicating deletion move.*
- #define VRNA_MOVESET_SHIFT 16

    *Option flag indicating shift move.*
- #define VRNA_MOVESET_NO_LP 32

    *Option flag indicating moves without lonely base pairs.*
- #define VRNA_MOVESET_DEFAULT (VRNA_MOVESET_INSERTION | VRNA_MOVESET_DELETION)

    *Option flag indicating default move set, i.e. insertions/deletion of a base pair.*

**Functions**

- void vrna_move_list_free (vrna_move_t ∗moves)
- void vrna_move_apply (short ∗pt, const vrna_move_t ∗m)

    *Apply a particular move / transition to a secondary structure, i.e. transform a structure.*
- void vrna_loopidx_update (int ∗loopidx, const short ∗pt, int length, const vrna_move_t ∗m)

    *Alters the loopIndices array that was constructed with vrna_loopidx_from_ptable().*
- vrna_move_t ∗ vrna_neighbors (vrna_fold_compound_t ∗vc, const short ∗pt, unsigned int options)

    *Generate neighbors of a secondary structure.*
- vrna_move_t ∗ vrna_neighbors_successive (const vrna_fold_compound_t ∗vc, const vrna_move_t ∗curr←
    _move, const short ∗prev_pt, const vrna_move_t ∗prev_neighbors, int size_prev_neighbors, int ∗size_←
    neighbors, unsigned int options)

    *Generate neighbors of a secondary structure (the fast way)*

### 17.51.1 Detailed Description

Methods to compute the neighbors of an RNA secondary structure.

## 17.52 ViennaRNA/params.h File Reference

Functions to deal with sets of energy parameters.

Include dependency graph for params.h:

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

**Data Structures**

- struct vrna_param_s

    *The datastructure that contains temperature scaled energy parameters. More...*
- struct vrna_exp_param_s

    *The data structure that contains temperature scaled Boltzmann weights of the energy parameters. More...*

## Typedefs

- typedef struct vrna_param_s vrna_param_t

  *Typename for the free energy parameter data structure vrna_params.*
- typedef struct vrna_exp_param_s vrna_exp_param_t

  *Typename for the Boltzmann factor data structure vrna_exp_params.*
- typedef struct vrna_param_s paramT

  *Old typename of vrna_param_s.*
- typedef struct vrna_exp_param_s pf_paramT

  *Old typename of vrna_exp_param_s.*

## Functions

- vrna_param_t ∗ vrna_params (vrna_md_t ∗md)

  *Get a data structure containing prescaled free energy parameters.*
- vrna_param_t ∗ vrna_params_copy (vrna_param_t ∗par)

  *Get a copy of the provided free energy parameters.*
- vrna_exp_param_t ∗ vrna_exp_params (vrna_md_t ∗md)

  *Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.*
- vrna_exp_param_t ∗ vrna_exp_params_comparative (unsigned int n_seq, vrna_md_t ∗md)

  *Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)*
- vrna_exp_param_t ∗ vrna_exp_params_copy (vrna_exp_param_t ∗par)

  *Get a copy of the provided free energy parameters (provided as Boltzmann factors)*
- void vrna_params_subst (vrna_fold_compound_t ∗vc, vrna_param_t ∗par)

  *Update/Reset energy parameters data structure within a vrna_fold_compound_t.*
- void vrna_exp_params_subst (vrna_fold_compound_t ∗vc, vrna_exp_param_t ∗params)

  *Update the energy parameters for subsequent partition function computations.*
- void vrna_exp_params_rescale (vrna_fold_compound_t ∗vc, double ∗mfe)

  *Rescale Boltzmann factors for partition function computations.*
- void vrna_params_reset (vrna_fold_compound_t ∗vc, vrna_md_t ∗md_p)

  *Reset free energy parameters within a vrna_fold_compound_t according to provided, or default model details.*
- void vrna_exp_params_reset (vrna_fold_compound_t ∗vc, vrna_md_t ∗md_p)

  *Reset Boltzmann factors for partition function computations within a vrna_fold_compound_t according to provided, or default model details.*
- vrna_exp_param_t ∗ get_scaled_pf_parameters (void)
- vrna_exp_param_t ∗ get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)

  *Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.*
- vrna_exp_param_t ∗ get_boltzmann_factor_copy (vrna_exp_param_t ∗parameters)

  *Get a copy of already precomputed Boltzmann factors.*
- vrna_exp_param_t ∗ get_scaled_alipf_parameters (unsigned int n_seq)

  *Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)*
- vrna_exp_param_t ∗ get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)

  *Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.*
- vrna_param_t ∗ scale_parameters (void)

  *Get precomputed energy contributions for all the known loop types.*
- vrna_param_t ∗ get_scaled_parameters (double temperature, vrna_md_t md)

  *Get precomputed energy contributions for all the known loop types.*

### 17.52.1 Detailed Description

Functions to deal with sets of energy parameters.

## 17.53 ViennaRNA/part_func.h File Reference

Partition function implementations.

Include dependency graph for part_func.h:

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

### Data Structures

- struct vrna_dimer_pf_s

    *Data structure returned by vrna_pf_dimer() More...*

### Typedefs

- typedef struct vrna_dimer_pf_s vrna_dimer_pf_t

    *Typename for the data structure that stores the dimer partition functions, vrna_dimer_pf_s, as returned by vrna_pf_dimer()*

- typedef struct vrna_dimer_pf_s cofoldF

    *Backward compatibility typedef for vrna_dimer_pf_s.*

### Functions

- float vrna_pf (vrna_fold_compound_t *vc, char *structure)

    *Compute the partition function $Q$ for a given RNA sequence, or sequence alignment.*

- float vrna_pf_fold (const char *sequence, char *structure, vrna_ep_t **pl)

    *Compute Partition function $Q$ (and base pair probabilities) for an RNA sequence using a comparative method.*

- float vrna_pf_circfold (const char *sequence, char *structure, vrna_ep_t **pl)

    *Compute Partition function $Q$ (and base pair probabilities) for a circular RNA sequences using a comparative method.*

- vrna_dimer_pf_t vrna_pf_dimer (vrna_fold_compound_t *vc, char *structure)

    *Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

- int vrna_pf_float_precision (void)

    *Find out whether partition function computations are using single precision floating points.*

- float pf_fold_par (const char *sequence, char *structure, vrna_exp_param_t *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, vrna_exp_param_t ∗parameters)

    *Recalculate energy parameters.*

- FLT_OR_DBL ∗ 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.*

- int get_pf_arrays (short ∗∗S_p, short ∗∗S1_p, char ∗∗ptype_p, FLT_OR_DBL ∗∗qb_p, FLT_OR_DBL ∗∗qm↩
    _p, FLT_OR_DBL ∗∗q1k_p, FLT_OR_DBL ∗∗qln_p)

    *Get the pointers to (almost) all relevant 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.*

- 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, FLT_OR_DBL ∗pr)

    *Get the mean base pair distance in the thermodynamic ensemble.*

- vrna_ep_t ∗ stackProb (double cutoff)

    *Get the probability of stacks.*

- void init_pf_fold (int length)

    *Allocate space for pf_fold()*

- char ∗ centroid (int length, double ∗dist)
- char ∗ get_centroid_struct_gquad_pr (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 back-tracking.*

## 17.53.1 Detailed Description

Partition function implementations.

This file includes (almost) all function declarations within the **RNAlib** that are related to Partion function computations

## 17.53.2 Function Documentation

**17.53.2.1 vrna_pf_float_precision()**

```
int vrna_pf_float_precision (
             void  )
```

Find out whether partition function computations are using single precision floating points.

**See also**

FLT_OR_DBL

**Returns**

1 if single precision is used, 0 otherwise

**17.53.2.2 stackProb()**

```
vrna_ep_t* stackProb (
             double cutoff )
```

Get the probability of stacks.

**Deprecated** Use vrna_stack_prob() instead!

**17.53.2.3 init_pf_fold()**

```
void init_pf_fold (
             int length )
```

Allocate space for pf_fold()

**Deprecated** This function is obsolete and will be removed soon!

**17.53.2.4 centroid()**

```
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()

**17.53.2.5 get_centroid_struct_gquad_pr()**

```
char* get_centroid_struct_gquad_pr (
            int length,
            double * dist )
```

**Deprecated** This function is deprecated and should not be used anymore as it is not threadsafe!

**See also**

> vrna_centroid(), vrna_centroid_from_probs(), vrna_centroid_from_plist()

**17.53.2.6 mean_bp_dist()**

```
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!

**17.53.2.7 expLoopEnergy()**

```
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

**17.53.2.8 expHairpinEnergy()**

```
double expHairpinEnergy (
            int u,
            int type,
            short si1,
            short sj1,
            const char * string )
```

**Deprecated** Use exp_E_Hairpin() from loop_energies.h instead

## 17.54 ViennaRNA/part_func_co.h File Reference

Partition function for two RNA sequences.

Include dependency graph for part_func_co.h:

### Functions

- vrna_dimer_pf_t vrna_pf_co_fold (const char ∗seq, char ∗structure, vrna_ep_t ∗∗pl)

   *Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*
- vrna_dimer_pf_t co_pf_fold (char ∗sequence, char ∗structure)

   *Calculate partition function and base pair probabilities.*
- vrna_dimer_pf_t co_pf_fold_par (char ∗sequence, char ∗structure, vrna_exp_param_t ∗parameters, int calculate_bppm, int is_constrained)

   *Calculate partition function and base pair probabilities.*
- vrna_ep_t ∗ get_plist (vrna_ep_t ∗pl, int length, double cut_off)
- void compute_probabilities (double FAB, double FEA, double FEB, vrna_ep_t ∗prAB, vrna_ep_t ∗prA, vrna_ep_t ∗prB, int Alength)

   *Compute Boltzmann probabilities of dimerization without homodimers.*
- void init_co_pf_fold (int length)
- FLT_OR_DBL ∗ 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, vrna_exp_param_t ∗parameters)

   *Recalculate energy parameters.*

### Variables

- int mirnatog

   *Toggles no intrabp in 2nd mol.*
- double F_monomer [2]

   *Free energies of the two monomers.*

### 17.54.1 Detailed Description

Partition function for two RNA sequences.

### 17.54.2 Function Documentation

**17.54.2.1 co_pf_fold()**

vrna_dimer_pf_t 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.

**Deprecated** {Use vrna_pf_dimer() instead!}

**Parameters**

| | |
|---|---|
| *sequence* | Concatenated RNA sequences |
| *structure* | Will hold the structure or constraints |

**Returns**

> vrna_dimer_pf_t structure containing a set of energies needed for concentration computations.

**17.54.2.2 co_pf_fold_par()**

vrna_dimer_pf_t co_pf_fold_par (
           char * *sequence,*
           char * *structure,*
           vrna_exp_param_t * *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.

**Deprecated** Use vrna_pf_dimer() instead!

**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**

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

**17.54.2.3 get_plist()**

```
vrna_ep_t* get_plist (
            vrna_ep_t * 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!

**17.54.2.4 compute_probabilities()**

```
void compute_probabilities (
            double FAB,
            double FEA,
            double FEB,
            vrna_ep_t * prAB,
            vrna_ep_t * prA,
            vrna_ep_t * 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.

**Deprecated** { Use vrna_pf_dimer_probs() instead!}

**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 |

**17.54.2.5   init_co_pf_fold()**

```
void init_co_pf_fold (
            int length )
```

DO NOT USE THIS FUNCTION ANYMORE

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

**17.54.2.6   export_co_bppm()**

```
FLT_OR_DBL* 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];
```

**Deprecated** This function is deprecated and will be removed soon! The base pair probability array is available through the vrna_fold_compound_t data structure, and its associated vrna_mx_pf_t member.

**See also**

vrna_idx_row_wise()

**Returns**

A pointer to the base pair probability array

---

**17.54.2.7 free_co_pf_arrays()**

```
void free_co_pf_arrays (
            void  )
```

Free the memory occupied by co_pf_fold()

**Deprecated** This function will be removed for the new API soon! See vrna_pf_dimer(), vrna_fold_compound(), and vrna_fold_compound_free() for an alternative

**17.54.2.8 update_co_pf_params()**

```
void update_co_pf_params (
            int length )
```

Recalculate energy parameters.

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

**Deprecated** Use vrna_exp_params_subst() instead!

**Parameters**

| length | Length of the current RNA sequence |
|--------|-------------------------------------|

**17.54.2.9 update_co_pf_params_par()**

```
void update_co_pf_params_par (
            int length,
            vrna_exp_param_t * 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.

**Deprecated** Use vrna_exp_params_subst() instead!

**Parameters**

| length | Length of the current RNA sequence |
|---|---|
| parameters | data structure containing the precomputed Boltzmann factors |

## 17.55 ViennaRNA/part_func_up.h File Reference

Implementations for accessibility and RNA-RNA interaction as a 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().*

### 17.55.1 Detailed Description

Implementations for accessibility and RNA-RNA interaction as a stepwise process.

## 17.56 ViennaRNA/perturbation_fold.h File Reference

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

Include dependency graph for perturbation_fold.h:

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

**Macros**

- #define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

  *Use the sum of squared aberrations as objective function.*
- #define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

  *Use the sum of absolute aberrations as objective function.*
- #define VRNA_MINIMIZER_DEFAULT 0

  *Use a custom implementation of the gradient descent algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_CONJUGATE_FR 1

  *Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_CONJUGATE_PR 2

  *Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_VECTOR_BFGS 3

  *Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_VECTOR_BFGS2 4

  *Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- #define VRNA_MINIMIZER_STEEPEST_DESCENT 5

  *Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.*

**Typedefs**

- typedef void(∗ progress_callback) (int iteration, double score, double ∗epsilon)

  *Callback for following the progress of the minimization process.*

**Functions**

- void vrna_sc_minimize_pertubation (vrna_fold_compound_t ∗vc, const double ∗q_prob_unpaired, int objective_function, double sigma_squared, double tau_squared, int algorithm, int sample_size, double ∗epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback)

  *Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.*

### 17.56.1 Detailed Description

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of neccessary adjustments.

## 17.57 VriennaRNA/plot_aln.h File Reference

Various functions for plotting Sequence / Structure Alignments.

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

**Functions**

- int PS_color_aln (const char ∗structure, const char ∗filename, const char ∗seqs[ ], const char ∗names[ ])

  *Produce PostScript sequence alignment color-annotated by consensus structure.*

- int aliPS_color_aln (const char ∗structure, const char ∗filename, const char ∗seqs[ ], const char ∗names[ ])

### 17.57.1  Detailed Description

Various functions for plotting Sequence / Structure Alignments.

## 17.58  ViennaRNA/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:

**Data Structures**

- struct COORDINATE

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

**Macros**

- #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.*

## 17.58.1 Detailed Description

Secondary structure plot layout algorithms.

## 17.59 ViennaRNA/plot_structure.h File Reference

Various functions for plotting RNA secondary structures.

Include dependency graph for plot_structure.h:

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

**Functions**

- int PS_rna_plot_snoop_a (const char ∗string, const char ∗structure, const char ∗ssfile, int ∗relative_access, const char ∗seqs[ ])
- int vrna_file_PS_rnaplot (const char ∗seq, const char ∗structure, const char ∗file, vrna_md_t ∗md_p)

  *Produce a secondary structure graph in PostScript and write it to 'filename'.*

- int vrna_file_PS_rnaplot_a (const char ∗seq, const char ∗structure, const char ∗file, const char ∗pre, const char ∗post, vrna_md_t ∗md_p)

  *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_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 PS_rna_plot_a_gquad (char ∗string, char ∗structure, char ∗ssfile, char ∗pre, char ∗post)

  *Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)*

### 17.59.1 Detailed Description

Various functions for plotting RNA secondary structures.

### 17.59.2 Function Documentation

#### 17.59.2.1 PS_rna_plot_snoop_a()

```
int PS_rna_plot_snoop_a (
            const char * string,
            const char * structure,
            const char * ssfile,
            int * relative_access,
            const char * seqs[] )
```

plotting_utils

#### 17.59.2.2 vrna_file_PS_rnaplot()

```
int vrna_file_PS_rnaplot (
            const char * seq,
            const char * structure,
            const char * file,
            vrna_md_t * md_p )
```

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**

| | |
|---|---|
| *seq* | The RNA sequence |
| *structure* | The secondary structure in dot-bracket notation |
| *file* | The filename of the postscript output |
| *md_p* | Model parameters used to generate a commandline option string in the output (Maybe NULL) |

**Returns**

    1 on success, 0 otherwise

#### 17.59.2.3 vrna_file_PS_rnaplot_a()

```
int vrna_file_PS_rnaplot_a (
            const char * seq,
```

```
              const char * structure,
              const char * file,
              const char * pre,
              const char * post,
              vrna_md_t * md_p )
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

Same as vrna_file_PS_rnaplot() 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**

| seq       | 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                         |
| md_p      | Model parameters used to generate a commandline option string in the output (Maybe NULL) |

**Returns**

> 1 on success, 0 otherwise

**17.59.2.4  gmlRNA()**

```
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

**17.59.2.5 ssv_rna_plot()**

```
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

**17.59.2.6 svg_rna_plot()**

```
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

**17.59.2.7 xrna_plot()**

```
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

**17.59.2.8   PS_rna_plot()**

```
int PS_rna_plot (
          char * string,
          char * structure,
          char * file )
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

**Deprecated** Use vrna_file_PS_rnaplot() instead!

**17.59.2.9   PS_rna_plot_a()**

```
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'.

**Deprecated** Use vrna_file_PS_rnaplot_a() instead!

**17.59.2.10  PS_rna_plot_a_gquad()**

```
int PS_rna_plot_a_gquad (
            char * string,
            char * structure,
            char * ssfile,
            char * pre,
            char * post )
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)

**Deprecated** Use vrna_file_PS_rnaplot_a() instead!

## 17.60  ViennaRNA/plot_utils.h File Reference

Various utilities to assist in plotting secondary structures and consensus structures.

### Functions

- char ** vrna_annotate_covar_struct (const char **alignment, const char *structure, vrna_md_t *md)

  *Produce covariance annotation for an alignment given a secondary structure.*
- vrna_cpair_t * vrna_annotate_covar_pairs (const char **alignment, vrna_ep_t *pl, vrna_ep_t *mfel, double threshold, vrna_md_t *md)

  *Produce covariance annotation for an alignment given a set of base pairs.*

### 17.60.1  Detailed Description

Various utilities to assist in plotting secondary structures and consensus structures.

## 17.61  ViennaRNA/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 (FLT_OR_DBL *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)

### 17.61.1 Function Documentation

#### 17.61.1.1 profile_edit_distance()

```
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

#### 17.61.1.2 Make_bp_profile_bppm()

```
float* Make_bp_profile_bppm (
            FLT_OR_DBL * 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

#### 17.61.1.3 free_profile()

```
void free_profile (
            float * T )
```

free space allocated in Make_bp_profile

Backward compatibility only. You can just use plain free()

**17.61.1.4  Make_bp_profile()**

```
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

## 17.62  ViennaRNA/PS_dot.h File Reference

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Include dependency graph for PS_dot.h:

**Data Structures**

- struct vrna_dotplot_auxdata_t

**Functions**

- 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 PS_dot_plot (char ∗string, char ∗file)

  *Produce postscript dot-plot.*

### 17.62.1  Detailed Description

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

## 17.63  ViennaRNA/read_epars.h File Reference

Read and write energy parameter files.

**Functions**

- const char ∗ last_parameter_file (void)

  *Get the file name of the parameter file that was most recently loaded.*

- 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.*

## 17.63.1 Detailed Description

Read and write energy parameter files.

## 17.64 ViennaRNA/ribo.h File Reference

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

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

**Functions**

- float ∗∗ get_ribosum (const char ∗∗Alseq, int n_seq, int length)

  *Retrieve a RiboSum Scoring Matrix for a given Alignment.*

- float ∗∗ readribosum (char ∗name)

  *Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.*

## 17.64.1 Detailed Description

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

## 17.65 ViennaRNA/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.*

### 17.65.1  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)
*
```

## 17.66 ViennaRNA/sequence.h File Reference

Functions and data structures related to sequence representations.

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

**Data Structures**

- struct vrna_sequence_s

    *Data structure representing a nucleotide sequence.*

- enum vrna_seq_type_e { VRNA_SEQ_UNKNOWN, VRNA_SEQ_RNA, VRNA_SEQ_DNA }

    *A enumerator used in vrna_sequence_s to distinguish different nucleotide sequences.*

- typedef struct vrna_sequence_s vrna_seq_t

    *Typename for nucleotide sequence representation data structure vrna_sequence_s.*

### 17.66.1 Detailed Description

Functions and data structures related to sequence representations.

### 17.66.2 Enumeration Type Documentation

#### 17.66.2.1 vrna_seq_type_e

```
enum vrna_seq_type_e
```

A enumerator used in vrna_sequence_s to distinguish different nucleotide sequences.

**Enumerator**

| VRNA_SEQ_UNKNOWN | Nucleotide sequence represents an Unkown type. |
|---|---|
| VRNA_SEQ_RNA | Nucleotide sequence represents an RNA type. |
| VRNA_SEQ_DNA | Nucleotide sequence represents a DNA type. |

## 17.67 ViennaRNA/stream_output.h File Reference

An implementation of a buffered, ordered stream output data structure.

**Typedefs**

- typedef struct vrna_ordered_stream_s ∗ vrna_ostream_t

    *An ordered output stream structure with unordered insert capabilities.*
- typedef void() vrna_callback_stream_output(void ∗auxdata, unsigned int i, void ∗data)

    *Ordered stream processing callback.*

**Functions**

- vrna_ostream_t vrna_ostream_init (vrna_callback_stream_output ∗output, void ∗auxdata)

    *Get an initialized ordered output stream.*
- void vrna_ostream_free (vrna_ostream_t dat)

    *Free an initialized ordered output stream.*
- void vrna_ostream_request (vrna_ostream_t dat, unsigned int num)

    *Request index in ordered output stream.*
- void vrna_ostream_provide (vrna_ostream_t dat, unsigned int i, void ∗data)

    *Provide output stream data for a particular index.*

### 17.67.1 Detailed Description

An implementation of a buffered, ordered stream output data structure.

## 17.68 ViennaRNA/string_utils.h File Reference

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

Include dependency graph for string_utils.h:

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

**Macros**

- #define XSTR(s) STR(s)

    *Stringify a macro after expansion.*
- #define STR(s) #s

    *Stringify a macro argument.*
- #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**

- char ∗ vrna_strdup_printf (const char ∗format,...)

    *Safely create a formatted string.*

- char ∗ vrna_strdup_vprintf (const char ∗format, va_list argp)

    *Safely create a formatted string.*

- int vrna_strcat_printf (char ∗∗dest, const char ∗format,...)

    *Safely append a formatted string to another string.*

- int vrna_strcat_vprintf (char ∗∗dest, const char ∗format, va_list args)

    *Safely append a formatted string to another string.*

- char ∗∗ vrna_strsplit (const char ∗string, const char ∗delimiter)

    *Split a string into tokens using a delimiting character.*

- char ∗ vrna_random_string (int l, const char symbols[ ])

    *Create a random string using characters from a specified symbol set.*

- int vrna_hamming_distance (const char ∗s1, const char ∗s2)

    *Calculate hamming distance between two sequences.*

- int vrna_hamming_distance_bound (const char ∗s1, const char ∗s2, int n)

    *Calculate hamming distance between two sequences up to a specified length.*

- void vrna_seq_toRNA (char ∗sequence)

    *Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.*

- void vrna_seq_toupper (char ∗sequence)

    *Convert an input sequence to uppercase.*

- char ∗ vrna_cut_point_insert (const char ∗string, int cp)

    *Add a separating '&' character into a string according to cut-point position.*

- char ∗ vrna_cut_point_remove (const char ∗string, int ∗cp)

    *Remove a separating '&' character from a string.*

- void str_uppercase (char ∗sequence)

    *Convert an input sequence to uppercase.*

- void str_DNA2RNA (char ∗sequence)

    *Convert a DNA input sequence to RNA alphabet.*

- 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.*

## 17.68.1 Detailed Description

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

## 17.68.2 Function Documentation

**17.68.2.1  str_uppercase()**

```
void str_uppercase (
            char * sequence )
```

Convert an input sequence to uppercase.

**Deprecated**  Use vrna_seq_toupper() instead!

**17.68.2.2  str_DNA2RNA()**

```
void str_DNA2RNA (
            char * sequence )
```

Convert a DNA input sequence to RNA alphabet.

**Deprecated**  Use vrna_seq_toRNA() instead!

**17.68.2.3  random_string()**

```
char* random_string (
            int l,
            const char symbols[] )
```

Create a random string using characters from a specified symbol set.

**Deprecated**  Use vrna_random_string() instead!

**17.68.2.4  hamming()**

```
int hamming (
            const char * s1,
            const char * s2 )
```

Calculate hamming distance between two sequences.

**Deprecated**  Use vrna_hamming_distance() instead!

**17.68.2.5 hamming_bound()**

```
int hamming_bound (
            const char * s1,
            const char * s2,
            int n )
```

Calculate hamming distance between two sequences up to a specified length.

**Deprecated** Use vrna_hamming_distance_bound() instead!

## 17.69 ViennaRNA/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.*

### 17.69.1 Detailed Description

Functions for String Alignment.

### 17.69.2 Function Documentation

**17.69.2.1 Make_swString()**

```
swString* Make_swString (
            char * string )
```

Convert a structure into a format suitable for string_edit_distance().

**Parameters**

| string | |
|--------|--|

**Returns**

**17.69.2.2  string_edit_distance()**

```
float string_edit_distance (
            swString * T1,
            swString * T2 )
```

Calculate the string edit distance of T1 and T2.

**Parameters**

| T1 | |
|----|----|
| T2 | |

**Returns**

# 17.70  ViennaRNA/structure_utils.h File Reference

Various utility- and helper-functions for secondary structure parsing, converting, etc.

Include dependency graph for structure_utils.h:

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

**Data Structures**

- struct vrna_hx_s

  *Data structure representing an entry of a helix list. More...*

- struct vrna_elem_prob_s

  *Data structure representing a single entry of an element probability list (e.g. list of pair probabilities) More...*

**Macros**

- #define [VRNA_BRACKETS_ALPHA](#) 4U

  *Bitflag to indicate secondary structure notations using uppercase/lowercase letters from the latin alphabet.*
- #define [VRNA_BRACKETS_RND](#) 8U

  *Bitflag to indicate secondary structure notations using round brackets (parenthesis), ( )*
- #define [VRNA_BRACKETS_CLY](#) 16U

  *Bitflag to indicate secondary structure notations using curly brackets, { }*
- #define [VRNA_BRACKETS_ANG](#) 32U

  *Bitflag to indicate secondary structure notations using angular brackets, <>*
- #define [VRNA_BRACKETS_SQR](#) 64U

  *Bitflag to indicate secondary structure notations using square brackets, [ ]*
- #define [VRNA_BRACKETS_DEFAULT](#)

  *Default bitmask to indicate secondary structure notation using any pair of brackets.*

**Typedefs**

- typedef struct [vrna_hx_s](#) [vrna_hx_t](#)

  *Convenience typedef for data structure [vrna_hx_s](#).*
- typedef struct [vrna_elem_prob_s](#) [vrna_ep_t](#)

  *Convenience typedef for data structure #vrna_pp_s.*

**Functions**

- char ∗ [vrna_db_pack](#) (const char ∗struc)

  *Pack secondary secondary structure, 5:1 compression using base 3 encoding.*
- char ∗ [vrna_db_unpack](#) (const char ∗packed)

  *Unpack secondary structure previously packed with [vrna_db_pack()](#)*
- short ∗ [vrna_ptable](#) (const char ∗structure)

  *Create a pair table of a secondary structure.*
- short ∗ [vrna_ptable_from_string](#) (const char ∗string, unsigned int options)

  *Create a pair table for a secondary structure string.*
- short ∗ [vrna_pt_pk_get](#) (const char ∗structure)

  *Create a pair table of a secondary structure (pseudo-knot version)*
- short ∗ [vrna_ptable_copy](#) (const short ∗pt)

  *Get an exact copy of a pair table.*
- short ∗ [vrna_pt_ali_get](#) (const char ∗structure)

  *Create a pair table of a secondary structure (snoop align version)*
- short ∗ [vrna_pt_snoop_get](#) (const char ∗structure)

  *Create a pair table of a secondary structure (snoop version)*
- int ∗ [vrna_loopidx_from_ptable](#) (const short ∗pt)

  *Get a loop index representation of a structure.*
- void [vrna_db_flatten](#) (char ∗structure, unsigned int options)

  *Substitute pairs of brackets in a string with parenthesis.*
- void [vrna_db_flatten_to](#) (char ∗string, const char target[3], unsigned int options)

  *Substitute pairs of brackets in a string with another type of pair characters.*
- char ∗ [vrna_db_from_ptable](#) (short ∗pt)

  *Convert a pair table into dot-parenthesis notation.*
- char ∗ [vrna_db_from_WUSS](#) (const char ∗wuss)

*Convert a WUSS annotation string to dot-bracket format.*

- int vrna_bp_distance (const char ∗str1, const char ∗str2)

    *Compute the "base pair" distance between two secondary structures s1 and s2.*

- unsigned int ∗ vrna_refBPcnt_matrix (const short ∗reference_pt, unsigned int turn)

    *Make a reference base pair count matrix.*

- unsigned int ∗ vrna_refBPdist_matrix (const short ∗pt1, const short ∗pt2, unsigned int turn)

    *Make a reference base pair distance matrix.*

- char ∗ vrna_db_from_probs (const FLT_OR_DBL ∗pr, unsigned int length)

    *Create a dot-bracket like structure string from base pair probability matrix.*

- char vrna_bpp_symbol (const float ∗x)

    *Get a pseudo dot bracket notation for a given probability information.*

- char ∗ vrna_db_from_bp_stack (vrna_bp_stack_t ∗bp, unsigned int length)

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

- vrna_ep_t ∗ vrna_plist (const char ∗struc, float pr)

    *Create a vrna_ep_t from a dot-bracket string.*

- vrna_ep_t ∗ vrna_plist_from_probs (vrna_fold_compound_t ∗vc, double cut_off)

    *Create a vrna_ep_t from base pair probability matrix.*

- char ∗ vrna_db_from_plist (vrna_ep_t ∗pairs, unsigned int n)

    *Convert a list of base pairs into dot-bracket notation.*

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

    *Create a vrna_ep_t from a dot-bracket string.*

- 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 bp_distance (const char ∗str1, const char ∗str2)

    *Compute the "base pair" distance between two secondary structures s1 and s2.*

- unsigned int ∗ make_referenceBP_array (short ∗reference_pt, unsigned int turn)

    *Make a reference base pair count matrix.*

- unsigned int ∗ compute_BPdifferences (short ∗pt1, short ∗pt2, unsigned int turn)

    *Make a reference base pair distance matrix.*

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

    *Create a vrna_ep_t from a probability matrix.*

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

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

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

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

- void bppm_to_structure (char ∗structure, FLT_OR_DBL ∗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.*

### 17.70.1 Detailed Description

Various utility- and helper-functions for secondary structure parsing, converting, etc.

## 17.71 ViennaRNA/structured_domains.h File Reference

This module provides interfaces that deal with additional structured domains in the folding grammar.

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

### Data Structures

- struct vrna_structured_domains_s

### 17.71.1 Detailed Description

This module provides interfaces that deal with additional structured domains in the folding grammar.

## 17.72 ViennaRNA/subopt.h File Reference

RNAsubopt and density of states declarations.

Include dependency graph for subopt.h:

### Data Structures

- struct vrna_subopt_sol_s

  *Solution element from subopt.c.*

### Macros

- #define MAXDOS 1000

  *Maximum density of states discretization for subopt.*

### Typedefs

- typedef struct vrna_subopt_sol_s vrna_subopt_solution_t

  *Typename for the subopt solution list repesenting data structure vrna_subopt_sol_s.*
- typedef void() vrna_subopt_callback(const char *stucture, float energy, void *data)

  *Callback for vrna_subopt_cb().*
- typedef struct vrna_subopt_sol_s SOLUTION

  *Backward compatibility typedef for vrna_subopt_sol_s.*

**Functions**

- vrna_subopt_solution_t ∗ vrna_subopt (vrna_fold_compound_t ∗vc, int delta, int sorted, FILE ∗fp)

  *Returns list of subopt structures or writes to fp.*

- void vrna_subopt_cb (vrna_fold_compound_t ∗vc, int delta, vrna_subopt_callback ∗cb, void ∗data)

  *Generate suboptimal structures within an energy band around the MFE.*

- vrna_subopt_solution_t ∗ vrna_subopt_zuker (vrna_fold_compound_t ∗vc)

  *Compute Zuker type suboptimal structures.*

- 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, vrna_param_t ∗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.*

- SOLUTION ∗ zukersubopt (const char ∗string)

  *Compute Zuker type suboptimal structures.*

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

  *Compute Zuker type suboptimal structures.*

**Variables**

- double print_energy

  *printing threshold for use with logML*

- int subopt_sorted

  *Sort output by energy.*

- int density_of_states [MAXDOS+1]

  *The Density of States.*

## 17.72.1  Detailed Description

RNAsubopt and density of states declarations.

## 17.72.2  Typedef Documentation

### 17.72.2.1  SOLUTION

```
typedef struct vrna_subopt_sol_s SOLUTION
```

Backward compatibility typedef for vrna_subopt_sol_s.

**Deprecated**  Use vrna_subopt_solution_t instead!

## 17.73 ViennaRNA/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.*

### 17.73.1 Detailed Description

Functions for Tree Edit Distances.

### 17.73.2 Function Documentation

#### 17.73.2.1 make_tree()

```
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**

**17.73.2.2 tree_edit_distance()**

```
float tree_edit_distance (
            Tree * T1,
            Tree * T2 )
```

Calculates the edit distance of the two trees.

**Parameters**

| T1 | |
|----|----|
| T2 | |

**Returns**

**17.73.2.3 free_tree()**

```
void free_tree (
            Tree * t )
```

Free the memory allocated for Tree t.

**Parameters**

| t | |
|---|---|

# 17.74 ViennaRNA/units.h File Reference

Physical Units and Functions to convert them into each other.

**Enumerations**

- enum vrna_unit_energy_e {
  VRNA_UNIT_J, VRNA_UNIT_KJ, VRNA_UNIT_CAL_IT, VRNA_UNIT_DACAL_IT,
  VRNA_UNIT_KCAL_IT, VRNA_UNIT_CAL, VRNA_UNIT_DACAL, VRNA_UNIT_KCAL,
  VRNA_UNIT_G_TNT, VRNA_UNIT_KG_TNT, VRNA_UNIT_T_TNT, VRNA_UNIT_EV,
  VRNA_UNIT_WH, VRNA_UNIT_KWH }

    *Energy / Work Units.*
- enum vrna_unit_temperature_e {
  VRNA_UNIT_K, VRNA_UNIT_DEG_C, VRNA_UNIT_DEG_F, VRNA_UNIT_DEG_R,
  VRNA_UNIT_DEG_N, VRNA_UNIT_DEG_DE, VRNA_UNIT_DEG_RE, VRNA_UNIT_DEG_RO }

    *Temperature Units.*

**Functions**

- double vrna_convert_energy (double energy, vrna_unit_energy_e from, vrna_unit_energy_e to)

  *Convert between energy / work units.*
- double vrna_convert_temperature (double temp, vrna_unit_temperature_e from, vrna_unit_temperature_e to)

  *Convert between temperature units.*

## 17.74.1  Detailed Description

Physical Units and Functions to convert them into each other.

## 17.75  ViennaRNA/unstructured_domains.h File Reference

Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.

Include dependency graph for unstructured_domains.h:

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

**Data Structures**

- struct vrna_unstructured_domain_s

  *Data structure to store all functionality for ligand binding. More...*
- struct vrna_unstructured_domain_motif_s

**Macros**

- #define VRNA_UNSTRUCTURED_DOMAIN_EXT_LOOP 1U

  *Flag to indicate ligand bound to unpaired stretch in the exterior loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_HP_LOOP 2U

  *Flag to indicate ligand bound to unpaired stretch in a hairpin loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_INT_LOOP 4U

  *Flag to indicate ligand bound to unpaired stretch in an interior loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_MB_LOOP 8U

  *Flag to indicate ligand bound to unpaired stretch in a multibranch loop.*
- #define VRNA_UNSTRUCTURED_DOMAIN_MOTIF 16U

  *Flag to indicate ligand binding without additional unbound nucleotides (motif-only)*
- #define VRNA_UNSTRUCTURED_DOMAIN_ALL_LOOPS

  *Flag to indicate ligand bound to unpaired stretch in any loop (convenience macro)*

**Typedefs**

- typedef struct vrna_unstructured_domain_s vrna_ud_t

    *Typename for the ligand binding extension data structure vrna_unstructured_domain_s.*
- typedef int() vrna_callback_ud_energy(vrna_fold_compound_t ∗vc, int i, int j, unsigned int loop_type, void ∗data)

    *Callback to retrieve binding free energy of a ligand bound to an unpaired sequence segment.*
- typedef FLT_OR_DBL() vrna_callback_ud_exp_energy(vrna_fold_compound_t ∗vc, int i, int j, unsigned int loop_type, void ∗data)

    *Callback to retrieve Boltzmann factor of the binding free energy of a ligand bound to an unpaired sequence segment.*
- typedef void() vrna_callback_ud_production(vrna_fold_compound_t ∗vc, void ∗data)

    *Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature.*
- typedef void() vrna_callback_ud_exp_production(vrna_fold_compound_t ∗vc, void ∗data)

    *Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature (partition function variant)*
- typedef void() vrna_callback_ud_probs_add(vrna_fold_compound_t ∗vc, int i, int j, unsigned int loop_type, FLT_OR_DBL exp_energy, void ∗data)

    *Callback to store/add equilibrium probability for a ligand bound to an unpaired sequence segment.*
- typedef FLT_OR_DBL() vrna_callback_ud_probs_get(vrna_fold_compound_t ∗vc, int i, int j, unsigned int loop_type, int motif, void ∗data)

    *Callback to retrieve equilibrium probability for a ligand bound to an unpaired sequence segment.*

**Functions**

- void vrna_ud_add_motif (vrna_fold_compound_t ∗vc, const char ∗motif, double motif_en, unsigned int loop←↩ _type)

    *Add an unstructured domain motif, e.g. for ligand binding.*
- int ∗ vrna_ud_get_motif_size_at (vrna_fold_compound_t ∗vc, int i, unsigned int loop_type)

    *Get a list of unique motif sizes that start at a certain position within the sequence.*
- void vrna_ud_remove (vrna_fold_compound_t ∗vc)

    *Remove ligand binding to unpaired stretches.*
- void vrna_ud_set_data (vrna_fold_compound_t ∗vc, void ∗data, vrna_callback_free_auxdata ∗free_cb)

    *Attach an auxiliary data structure.*
- void vrna_ud_set_prod_rule_cb (vrna_fold_compound_t ∗vc, vrna_callback_ud_production ∗pre_cb, vrna_callback_ud_energy ∗e_cb)

    *Attach production rule callbacks for free energies computations.*
- void vrna_ud_set_exp_prod_rule_cb (vrna_fold_compound_t ∗vc, vrna_callback_ud_exp_production ∗pre←↩ _cb, vrna_callback_ud_exp_energy ∗exp_e_cb)

    *Attach production rule for partition function.*
- void vrna_ud_set_prob_cb (vrna_fold_compound_t ∗vc, vrna_callback_ud_probs_add ∗setter, vrna_callback_ud_probs_get ∗getter)

**17.75.1 Detailed Description**

Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.

**17.75.2 Function Documentation**

**17.75.2.1 vrna_ud_set_prob_cb()**

```
void vrna_ud_set_prob_cb (
            vrna_fold_compound_t * vc,
            vrna_callback_ud_probs_add * setter,
            vrna_callback_ud_probs_get * getter )
```

**SWIG Wrapper Notes**  This function is attached as method **ud_set_prob_cb()** to objects of type *fold_compound*

## 17.76  ViennaRNA/utils.h File Reference

General utility- and helper-functions used throughout the *ViennaRNA Package*.

Include dependency graph for utils.h:

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

**Macros**

- #define VRNA_INPUT_ERROR 1U

  *Output flag of get_input_line(): "An ERROR has occured, maybe EOF".*
- #define VRNA_INPUT_QUIT 2U

  *Output flag of get_input_line(): "the user requested quitting the program".*
- #define VRNA_INPUT_MISC 4U

  *Output flag of get_input_line(): "something was read".*
- #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.*
- #define VRNA_INPUT_CONSTRAINT 32U

  *Input flag for get_input_line():*
  *Tell get_input_line() that we assume to read a structure constraint.*
- #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".*
- #define VRNA_INPUT_NO_REST 512U

  *Input switch for vrna_file_fasta_read_record(): "do fill rest array".*
- #define VRNA_INPUT_NO_SPAN 1024U

  *Input switch for vrna_file_fasta_read_record(): "never allow data to span more than one line".*
- #define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U

  *Input switch for vrna_file_fasta_read_record(): "do not skip empty lines".*
- #define VRNA_INPUT_BLANK_LINE 4096U

  *Output flag for vrna_file_fasta_read_record(): "read an empty line".*
- #define VRNA_INPUT_NOSKIP_COMMENTS 128U

  *Input switch for get_input_line(): "do not skip comment lines".*
- #define VRNA_INPUT_COMMENT 8192U

  *Output flag for vrna_file_fasta_read_record(): "read a comment".*

- #define MIN2(A, B) ((A) < (B) ? (A) : (B))

    *Get the minimum of two comparable values.*
- #define MAX2(A, B) ((A) > (B) ? (A) : (B))

    *Get the maximum of two comparable values.*
- #define MIN3(A, B, C) (MIN2( (MIN2((A),(B))) ,(C)))

    *Get the minimum of three comparable values.*
- #define MAX3(A, B, C) (MAX2( (MAX2((A),(B))) ,(C)))

    *Get the maximum of three comparable values.*

## Functions

- void ∗ vrna_alloc (unsigned size)

    *Allocate space safely.*
- void ∗ vrna_realloc (void ∗p, unsigned size)

    *Reallocate space safely.*
- void vrna_message_error (const char ∗format,...)

    *Print an error message and die.*
- void vrna_message_verror (const char ∗format, va_list args)

    *Print an error message and die.*
- void vrna_message_warning (const char ∗format,...)

    *Print a warning message.*
- void vrna_message_vwarning (const char ∗format, va_list args)

    *Print a warning message.*
- void vrna_message_info (FILE ∗fp, const char ∗format,...)

    *Print an info message.*
- void vrna_message_vinfo (FILE ∗fp, const char ∗format, va_list args)

    *Print an info message.*
- void vrna_init_rand (void)

    *Initialize seed for random number generator.*
- double vrna_urn (void)

    *get a random number from [0..1]*
- int vrna_int_urn (int from, int to)

    *Generates a pseudo random integer in a specified range.*
- char ∗ vrna_time_stamp (void)

    *Get a timestamp.*
- unsigned int get_input_line (char ∗∗string, unsigned int options)
- void vrna_message_input_seq_simple (void)

    *Print a line to stdout that asks for an input sequence.*
- void vrna_message_input_seq (const char ∗s)

    *Print a line with a user defined string and a ruler to stdout.*
- int ∗ vrna_idx_row_wise (unsigned int length)

    *Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.*
- int ∗ vrna_idx_col_wise (unsigned int length)

    *Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.*
- char ∗ get_line (FILE ∗fp)

    *Read a line of arbitrary length from a stream.*
- 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 [warn_user](const char message[ ])

    *Print a warning message.*
- void [nrerror](const char message[ ])

    *Die with an error message.*
- void ∗ [space](unsigned size)

    *Allocate space safely.*
- void ∗ [xrealloc](void ∗p, unsigned size)

    *Reallocate space safely.*
- 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.*
- void [filecopy](FILE ∗from, FILE ∗to)

    *Inefficient* `cp`
- char ∗ [time_stamp](void)

    *Get a timestamp.*

## Variables

- unsigned short [xsubi](3)

    *Current 48 bit random number.*

## 17.76.1  Detailed Description

General utility- and helper-functions used throughout the *ViennaRNA Package*.

## 17.76.2  Function Documentation

### 17.76.2.1  get_line()

```
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.

**Deprecated**  Use [vrna_read_line()](#) as a substitute!

**Parameters**

| | |
|---|---|
| *fp* | A file pointer to the stream where the function should read from |

**Returns**

A pointer to the resulting string

**17.76.2.2 print_tty_input_seq()**

```
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

**Deprecated** Use vrna_message_input_seq_simple() instead!

**17.76.2.3 print_tty_input_seq_str()**

```
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

**Deprecated** Use vrna_message_input_seq() instead!

**17.76.2.4 warn_user()**

```
void warn_user (
            const char message[] )
```

Print a warning message.

Print a warning message to *stderr*

**Deprecated** Use vrna_message_warning() instead!

**17.76.2.5 nrerror()**

```
void nrerror (
            const char message[] )
```

Die with an error message.

**Deprecated** Use vrna_message_error() instead!

**17.76.2.6 space()**

```
void* space (
            unsigned size )
```

Allocate space safely.

**Deprecated** Use vrna_alloc() instead!

**17.76.2.7 xrealloc()**

```
void* xrealloc (
            void * p,
            unsigned size )
```

Reallocate space safely.

**Deprecated** Use vrna_realloc() instead!

**17.76.2.8 init_rand()**

```
void init_rand (
            void  )
```

Make random number seeds.

**Deprecated** Use vrna_init_rand() instead!

**17.76.2.9  urn()**

```
double urn (
             void  )
```

get a random number from [0..1]

**Deprecated**  Use vrna_urn() instead!

**17.76.2.10  int_urn()**

```
int int_urn (
             int from,
             int to )
```

Generates a pseudo random integer in a specified range.

**Deprecated**  Use vrna_int_urn() instead!

**17.76.2.11  filecopy()**

```
void filecopy (
             FILE * from,
             FILE * to )
```

Inefficient `cp`

**Deprecated**  Use vrna_file_copy() instead!

**17.76.2.12  time_stamp()**

```
char* time_stamp (
             void  )
```

Get a timestamp.

**Deprecated**  Use vrna_time_stamp() instead!

## 17.77 ViennaRNA/walk.h File Reference

Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.

Include dependency graph for walk.h:

### Macros

- #define VRNA_PATH_STEEPEST_DESCENT 128

    *Option flag to request a steepest descent / gradient path.*
- #define VRNA_PATH_RANDOM 256

    *Option flag to request a random walk path.*
- #define VRNA_PATH_NO_TRANSITION_OUTPUT 512

    *Option flag to omit returning the transition path.*
- #define VRNA_PATH_DEFAULT (VRNA_PATH_STEEPEST_DESCENT | VRNA_MOVESET_DEFAULT)

    *Option flag to request defaults (steepest descent / default move set)*

### Functions

- vrna_move_t ∗ vrna_path (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int steps, unsigned int options)

    *Compute a path, store the final structure, and return a list of transition moves from the start to the final structure.*
- vrna_move_t ∗ vrna_path_gradient (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int options)

    *Compute a steepest descent / gradient path, store the final structure, and return a list of transition moves from the start to the final structure.*
- vrna_move_t ∗ vrna_path_random (vrna_fold_compound_t ∗vc, short ∗pt, unsigned int steps, unsigned int options)

    *Generate a random walk / path of a given length, store the final structure, and return a list of transition moves from the start to the final structure.*

### 17.77.1 Detailed Description

Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.

# Bibliography

[1] S.H. Bernhart, I.L. Hofacker, S. Will, A.R. Gruber, and P.F. Stadler. RNAalifold: Improved consensus structure prediction for RNA alignments. *BMC bioinformatics*, 9(1):474, 2008. 26

[2] S.H. Bernhart, H. Tafer, U. Mückstein, C. Flamm, P.F. Stadler, and I.L. Hofacker. Partition function and base pairing probabilities of RNA heterodimers. *Algorithms for Molecular Biology*, 1(1):3, 2006. 402

[3] R.E. Bruccoleri and G. Heinrich. An improved algorithm for nucleic acid secondary structure display. *Computer applications in the biosciences: CABIOS*, 4(1):167–173, 1988. 534

[4] Katherine E. Deigan, Tian W. Li, David H. Mathews, and Kevin M. Weeks. Accurate SHAPE-directed RNA structure determination. *PNAS*, 106:97–102, 2009. 350

[5] Christoph Flamm, Ivo L Hofacker, Sebastian Maurer-Stroh, Peter F Stadler, and Martin Zehl. Design of multi-stable RNA molecules. *RNA*, 7(02):254–265, 2001. 454, 455, 456, 457

[6] W. Fontana, P.F. Stadler, E.G. Bornberg-Bauer, T. Griesmacher, I.L. Hofacker, M. Tacker, P. Tarazona, E.D. Weinberger, and P. Schuster. RNA folding and combinatory landscapes. *Physical review E*, 47(3):2083, 1993. 29

[7] I.L. Hofacker, M. Fekete, and P.F. Stadler. Secondary structure prediction for aligned RNA sequences. *Journal of molecular biology*, 319(5):1059–1066, 2002. 26

[8] I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994. 1

[9] I.L. Hofacker and P.F. Stadler. Memory efficient folding algorithms for circular RNA secondary structures. *Bioinformatics*, 22(10):1172–1176, 2006. 24, 274, 306, 310, 388

[10] Ronny Lorenz, Stephan H. Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011. 1

[11] Ronny Lorenz, Christoph Flamm, and Ivo L. Hofacker. 2d projections of RNA folding landscapes. In Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, and Peter F. Stadler, editors, *German Conference on Bioinformatics 2009*, volume 157 of *Lecture Notes in Informatics*, pages 11–20, Bonn, September 2009. Gesellschaft f. Informatik. 27, 317

[12] Ronny Lorenz, Ivo L. Hofacker, and Peter F. Stadler. RNA folding with hard and soft constraints. *Algorithms for Molecular Biology*, 11(1):1–13, 2016. 22

[13] Ronny Lorenz, Dominik Luntzer, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Shape directed rna folding. *Bioinformatics*, 32(1):145–147, 2016. 349

[14] D.H. Mathews, M.D. Disney, J.L. Childs, S.J. Schroeder, M. Zuker, and D.H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287, 2004. 423

[15] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. 24, 25

[16] Joe Sawada. A fast algorithm to generate necklaces with fixed content. *Theoretical Computer Science*, 301(1):477–489, 2003. 377

[17] B.A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer applications in the biosciences: CABIOS*, 4(3):387–393, 1988. 29

[18] B.A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer applications in the biosciences: CABIOS*, 6(4):309–318, 1990. 17

[19] D.H. Turner and D.H. Mathews. NNDB: The nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Research*, 38(suppl 1):D280–D282, 2010. 423

[20] Stefan Washietl, Ivo L. Hofacker, Peter F. Stadler, and Manolis Kellis. RNA folding with soft constraints: reconciliation of probing data and thermodynamics secondary structure prediction. *Nucleic Acids Research*, 40(10):4261–4272, 2012. 356

[21] S. Wuchty, W. Fontana, I. L. Hofacker, and P. Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49(2):145–165, February 1999. 287, 288

[22] Kourosh Zarringhalam, Michelle M. Meyer, Ivan Dotu, Jeffrey H. Chuang, and Peter Clote. Integrating chemical footprinting data into RNA secondary structure prediction. *PLOS ONE*, 7(10), 2012. 351

[23] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244(4900):48–52, April 1989. 284

[24] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981. 24

# Index