

RNAlib-2.1.2

Generated by Doxygen 1.6.1

Tue Apr 11 12:10:08 2017



# Contents

<b>1</b>	<b>ViennaRNA Package core - RNAlib</b>	<b>1</b>
1.1	Introduction . . . . .	1
<b>2</b>	<b>Parsing and Comparing - Functions to Manipulate Structures</b>	<b>3</b>
<b>3</b>	<b>Utilities - Odds and Ends</b>	<b>9</b>
3.1	Producing secondary structure graphs . . . . .	10
3.2	Producing (colored) dot plots for base pair probabilities . . . . .	11
3.3	Producing (colored) alignments . . . . .	12
3.4	RNA sequence related utilities . . . . .	12
3.5	RNA secondary structure related utilities . . . . .	12
3.6	Miscellaneous Utilities . . . . .	13
<b>4</b>	<b>Example - A Small Example Program</b>	<b>15</b>
<b>5</b>	<b>Deprecated List</b>	<b>19</b>
<b>6</b>	<b>Module Index</b>	<b>23</b>
6.1	Modules . . . . .	23
<b>7</b>	<b>Data Structure Index</b>	<b>25</b>
7.1	Data Structures . . . . .	25
<b>8</b>	<b>File Index</b>	<b>27</b>
8.1	File List . . . . .	27
<b>9</b>	<b>Module Documentation</b>	<b>31</b>
9.1	RNA Secondary Structure Folding . . . . .	31
9.1.1	Detailed Description . . . . .	32

9.2	Calculating Minimum Free Energy (MFE) Structures . . . . .	33
9.2.1	Detailed Description . . . . .	34
9.2.2	Function Documentation . . . . .	34
9.2.2.1	fold_par . . . . .	34
9.2.2.2	fold . . . . .	35
9.2.2.3	circfold . . . . .	36
9.3	Calculating Partition Functions and Pair Probabilities . . . . .	37
9.3.1	Detailed Description . . . . .	38
9.3.2	Function Documentation . . . . .	39
9.3.2.1	pf_fold_par . . . . .	39
9.3.2.2	pf_fold . . . . .	40
9.3.2.3	pf_circ_fold . . . . .	41
9.3.2.4	free_pf_arrays . . . . .	41
9.3.2.5	update_pf_params . . . . .	42
9.3.2.6	export_bppm . . . . .	42
9.3.2.7	assign_plist_from_pr . . . . .	42
9.3.2.8	get_pf_arrays . . . . .	43
9.3.2.9	mean_bp_distance . . . . .	43
9.3.2.10	mean_bp_distance_pr . . . . .	44
9.4	Compute the structure with maximum expected accuracy (MEA) . . . . .	45
9.5	Compute the centroid structure . . . . .	46
9.5.1	Function Documentation . . . . .	46
9.5.1.1	get_centroid_struct_pl . . . . .	46
9.5.1.2	get_centroid_struct_pr . . . . .	46
9.6	Enumerating Suboptimal Structures . . . . .	48
9.7	Suboptimal structures according to Zuker et al. 1989 . . . . .	49
9.7.1	Function Documentation . . . . .	49
9.7.1.1	zukersubopt . . . . .	49
9.8	Suboptimal structures within an energy band around the MFE . . . . .	50
9.8.1	Function Documentation . . . . .	50
9.8.1.1	subopt . . . . .	50
9.8.1.2	subopt_circ . . . . .	51
9.9	Stochastic backtracking in the Ensemble . . . . .	52
9.9.1	Function Documentation . . . . .	52

---

9.9.1.1	<code>pbacktrack</code>	52
9.9.1.2	<code>pbacktrack_circ</code>	53
9.9.2	Variable Documentation	53
9.9.2.1	<code>st_back</code>	53
9.10	Calculate Secondary Structures of two RNAs upon Dimerization	54
9.10.1	Detailed Description	54
9.11	MFE Structures of two hybridized Sequences	55
9.11.1	Function Documentation	55
9.11.1.1	<code>cofold</code>	55
9.11.1.2	<code>export_cofold_arrays_gq</code>	56
9.11.1.3	<code>export_cofold_arrays</code>	56
9.12	Partition Function for two hybridized Sequences	58
9.12.1	Detailed Description	59
9.12.2	Function Documentation	59
9.12.2.1	<code>co_pf_fold</code>	59
9.12.2.2	<code>co_pf_fold_par</code>	60
9.12.2.3	<code>export_co_bppm</code>	60
9.12.2.4	<code>update_co_pf_params</code>	60
9.12.2.5	<code>update_co_pf_params_par</code>	61
9.12.2.6	<code>compute_probabilities</code>	61
9.12.2.7	<code>get_concentrations</code>	62
9.13	Partition Function for two hybridized Sequences as a stepwise Process	63
9.13.1	Detailed Description	63
9.13.2	Function Documentation	63
9.13.2.1	<code>pf_unstru</code>	63
9.13.2.2	<code>pf_interact</code>	64
9.14	Predicting Consensus Structures from Alignment(s)	66
9.14.1	Detailed Description	67
9.14.2	Function Documentation	67
9.14.2.1	<code>get_mpi</code>	67
9.14.2.2	<code>energy_of_alistruct</code>	68
9.14.2.3	<code>encode_ali_sequence</code>	68
9.14.2.4	<code>alloc_sequence_arrays</code>	68
9.14.2.5	<code>free_sequence_arrays</code>	69

9.14.2.6	get_alipf_arrays	69
9.14.3	Variable Documentation	70
9.14.3.1	cv_fact	70
9.14.3.2	nc_fact	70
9.15	MFE Consensus Structures for Sequence Alignment(s)	71
9.15.1	Function Documentation	71
9.15.1.1	alifold	71
9.15.1.2	circularifold	71
9.16	Partition Function and Base Pair Probabilities for Sequence Alignment(s)	73
9.16.1	Function Documentation	73
9.16.1.1	alipf_fold_par	73
9.16.1.2	alipf_fold	74
9.16.1.3	alipf_circ_fold	74
9.16.1.4	export_ali_bppm	74
9.17	Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	75
9.17.1	Function Documentation	75
9.17.1.1	alipbacktrack	75
9.18	Predicting Locally stable structures of large sequences	76
9.18.1	Detailed Description	76
9.19	Local MFE structure Prediction and Z-scores	77
9.19.1	Function Documentation	77
9.19.1.1	Lfold	77
9.19.1.2	Lfoldz	77
9.20	Partition functions for locally stable secondary structures	78
9.20.1	Function Documentation	78
9.20.1.1	update_pf_paramsLP	78
9.20.1.2	pfl_fold	78
9.20.1.3	putoutpU_prob	79
9.20.1.4	putoutpU_prob_bin	80
9.21	Local MFE consensus structures for Sequence Alignments	81
9.21.1	Function Documentation	81
9.21.1.1	aliLfold	81
9.22	Change and Precalculate Energy Parameter Sets and Boltzmann Factors	82

---

9.22.1	Detailed Description	83
9.22.2	Function Documentation	83
9.22.2.1	scale_parameters	83
9.22.2.2	get_scaled_parameters	83
9.22.2.3	get_scaled_pf_parameters	84
9.22.2.4	get_boltzmann_factors	84
9.22.2.5	get_boltzmann_factor_copy	84
9.23	Reading/Writing energy parameter sets from/to File	86
9.23.1	Detailed Description	86
9.23.2	Function Documentation	86
9.23.2.1	read_parameter_file	86
9.23.2.2	write_parameter_file	87
9.24	Converting energy parameter files	88
9.24.1	Detailed Description	89
9.24.2	Define Documentation	89
9.24.2.1	VRNA_CONVERT_OUTPUT_ALL	89
9.24.2.2	VRNA_CONVERT_OUTPUT_HP	89
9.24.2.3	VRNA_CONVERT_OUTPUT_STACK	89
9.24.2.4	VRNA_CONVERT_OUTPUT_MM_HP	89
9.24.2.5	VRNA_CONVERT_OUTPUT_MM_INT	89
9.24.2.6	VRNA_CONVERT_OUTPUT_MM_INT_1N	89
9.24.2.7	VRNA_CONVERT_OUTPUT_MM_INT_23	89
9.24.2.8	VRNA_CONVERT_OUTPUT_MM_MULTI	89
9.24.2.9	VRNA_CONVERT_OUTPUT_MM_EXT	89
9.24.2.10	VRNA_CONVERT_OUTPUT_DANGLES5	90
9.24.2.11	VRNA_CONVERT_OUTPUT_DANGLE3	90
9.24.2.12	VRNA_CONVERT_OUTPUT_INT_11	90
9.24.2.13	VRNA_CONVERT_OUTPUT_INT_21	90
9.24.2.14	VRNA_CONVERT_OUTPUT_INT_22	90
9.24.2.15	VRNA_CONVERT_OUTPUT_BULGE	90
9.24.2.16	VRNA_CONVERT_OUTPUT_INT	90
9.24.2.17	VRNA_CONVERT_OUTPUT_ML	90
9.24.2.18	VRNA_CONVERT_OUTPUT_MISC	90
9.24.2.19	VRNA_CONVERT_OUTPUT_SPECIAL_HP	90

9.24.2.20	VRNA_CONVERT_OUTPUT_VANILLA . . . . .	91
9.24.2.21	VRNA_CONVERT_OUTPUT_NINIO . . . . .	91
9.24.2.22	VRNA_CONVERT_OUTPUT_DUMP . . . . .	91
9.24.3	Function Documentation . . . . .	91
9.24.3.1	convert_parameter_file . . . . .	91
9.25	Energy evaluation . . . . .	93
9.25.1	Detailed Description . . . . .	93
9.25.2	Function Documentation . . . . .	94
9.25.2.1	energy_of_structure . . . . .	94
9.25.2.2	energy_of_struct_par . . . . .	94
9.25.2.3	energy_of_circ_structure . . . . .	95
9.25.2.4	energy_of_circ_struct_par . . . . .	95
9.25.2.5	energy_of_structure_pt . . . . .	96
9.25.2.6	energy_of_struct_pt_par . . . . .	96
9.26	Searching Sequences for Predefined Structures . . . . .	97
9.26.1	Detailed Description . . . . .	97
9.26.2	Function Documentation . . . . .	97
9.26.2.1	inverse_fold . . . . .	97
9.26.2.2	inverse_pf_fold . . . . .	98
9.26.3	Variable Documentation . . . . .	98
9.26.3.1	final_cost . . . . .	98
9.26.3.2	give_up . . . . .	98
9.26.3.3	inv_verbose . . . . .	98
9.27	Classified Dynamic Programming . . . . .	99
9.28	Distance based partitioning of the Secondary Structure Space . . . . .	100
9.28.1	Detailed Description . . . . .	100
9.29	Calculating MFE representatives of a Distance Based Partitioning . . . . .	101
9.29.1	Detailed Description . . . . .	101
9.29.2	Function Documentation . . . . .	102
9.29.2.1	get_TwoDfold_variables . . . . .	102
9.29.2.2	destroy_TwoDfold_variables . . . . .	102
9.29.2.3	TwoDfoldList . . . . .	102
9.29.2.4	TwoDfold_backtrack_f5 . . . . .	103
9.30	Calculate Partition Functions of a Distance Based Partitioning . . . . .	104



---

9.30.1 Detailed Description . . . . .	104
9.30.2 Function Documentation . . . . .	105
9.30.2.1 get_TwoDpfold_variables . . . . .	105
9.30.2.2 get_TwoDpfold_variables_from_MFE . . . . .	105
9.30.2.3 destroy_TwoDpfold_variables . . . . .	105
9.30.2.4 TwoDpfoldList . . . . .	106
9.31 Stochastic Backtracking of Structures from Distance Based Partitioning	107
9.31.1 Detailed Description . . . . .	107
9.31.2 Function Documentation . . . . .	107
9.31.2.1 TwoDpfold_pbacktrack . . . . .	107
9.31.2.2 TwoDpfold_pbacktrack5 . . . . .	108
9.32 Compute the Density of States . . . . .	109
9.32.1 Variable Documentation . . . . .	109
9.32.1.1 density_of_states . . . . .	109
9.33 Parsing and Comparing - Functions to Manipulate Structures . . . . .	110
<b>10 Data Structure Documentation</b>	<b>111</b>
10.1 bondT Struct Reference . . . . .	111
10.1.1 Detailed Description . . . . .	111
10.2 bondTEn Struct Reference . . . . .	112
10.2.1 Detailed Description . . . . .	112
10.3 cofoldF Struct Reference . . . . .	113
10.4 ConcEnt Struct Reference . . . . .	114
10.5 constrain Struct Reference . . . . .	115
10.5.1 Detailed Description . . . . .	115
10.6 COORDINATE Struct Reference . . . . .	116
10.6.1 Detailed Description . . . . .	116
10.7 cpair Struct Reference . . . . .	117
10.7.1 Detailed Description . . . . .	117
10.8 duplexT Struct Reference . . . . .	118
10.9 dupVar Struct Reference . . . . .	119
10.10folden Struct Reference . . . . .	120
10.11interact Struct Reference . . . . .	121
10.12intermediate_t Struct Reference . . . . .	122

---

10.13	INTERVAL Struct Reference	123
10.13.1	Detailed Description	123
10.14	LIST Struct Reference	124
10.15	LST_BUCKET Struct Reference	125
10.16	model_detailsT Struct Reference	126
10.16.1	Detailed Description	126
10.16.2	Field Documentation	126
10.16.2.1	dangles	126
10.17	move_t Struct Reference	128
10.18	PAIR Struct Reference	129
10.18.1	Detailed Description	129
10.19	pair_info Struct Reference	130
10.19.1	Detailed Description	130
10.20	pairpro Struct Reference	131
10.21	paramT Struct Reference	132
10.21.1	Detailed Description	132
10.22	path_t Struct Reference	133
10.23	pf_paramT Struct Reference	134
10.23.1	Detailed Description	134
10.23.2	Field Documentation	134
10.23.2.1	alpha	134
10.24	plist Struct Reference	136
10.24.1	Detailed Description	136
10.25	Postorder_list Struct Reference	137
10.26	pu_contrib Struct Reference	138
10.26.1	Detailed Description	138
10.27	pu_out Struct Reference	139
10.27.1	Detailed Description	139
10.28	sect Struct Reference	140
10.28.1	Detailed Description	140
10.29	snoopT Struct Reference	141
10.30	SOLUTION Struct Reference	142
10.30.1	Detailed Description	142
10.31	struct_en Struct Reference	143

---

10.32svm_model Struct Reference . . . . .	144
10.33swString Struct Reference . . . . .	145
10.34Tree Struct Reference . . . . .	146
10.35TwoDfold_solution Struct Reference . . . . .	147
10.35.1 Detailed Description . . . . .	147
10.36TwoDfold_vars Struct Reference . . . . .	148
10.36.1 Detailed Description . . . . .	149
10.37TwoDpfold_solution Struct Reference . . . . .	150
10.37.1 Detailed Description . . . . .	150
10.38TwoDpfold_vars Struct Reference . . . . .	151
10.38.1 Detailed Description . . . . .	152
<b>11 File Documentation</b>	<b>153</b>
11.1 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/2Dfold.h File Reference . . . . .	153
11.1.1 Detailed Description . . . . .	154
11.2 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/2Dpfold.h File Reference . . . . .	155
11.2.1 Detailed Description . . . . .	155
11.3 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/alifold.h File Reference . . . . .	156
11.3.1 Detailed Description . . . . .	157
11.3.2 Function Documentation . . . . .	158
11.3.2.1 update_alifold_params . . . . .	158
11.4 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/cofold.h File Reference . . . . .	159
11.4.1 Detailed Description . . . . .	160
11.4.2 Function Documentation . . . . .	160
11.4.2.1 get_monomere_mfes . . . . .	160
11.4.2.2 initialize_cofold . . . . .	160
11.5 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/convert_ epars.h File Reference . . . . .	161
11.5.1 Detailed Description . . . . .	161
11.6 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_ structures.h File Reference . . . . .	162
11.6.1 Detailed Description . . . . .	164

11.7	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist_ vars.h File Reference . . . . .	165
11.7.1	Detailed Description . . . . .	165
11.7.2	Variable Documentation . . . . .	165
11.7.2.1	edit_backtrack . . . . .	165
11.7.2.2	cost_matrix . . . . .	166
11.8	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/duplex.h File Reference . . . . .	167
11.8.1	Detailed Description . . . . .	167
11.9	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/edit_ cost.h File Reference . . . . .	168
11.9.1	Detailed Description . . . . .	168
11.10	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/energy_ const.h File Reference . . . . .	169
11.10.1	Detailed Description . . . . .	169
11.10.2	Define Documentation . . . . .	169
11.10.2.1	GASCONST . . . . .	169
11.10.2.2	K0 . . . . .	169
11.10.2.3	INF . . . . .	169
11.10.2.4	FORBIDDEN . . . . .	170
11.10.2.5	BONUS . . . . .	170
11.10.2.6	NBPAIRS . . . . .	170
11.10.2.7	TURN . . . . .	170
11.10.2.8	MAXLOOP . . . . .	170
11.11	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/findpath.h File Reference . . . . .	171
11.11.1	Detailed Description . . . . .	171
11.11.2	Function Documentation . . . . .	171
11.11.2.1	find_saddle . . . . .	171
11.11.2.2	get_path . . . . .	172
11.11.2.3	free_path . . . . .	172
11.12	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h File Reference . . . . .	173
11.12.1	Detailed Description . . . . .	175
11.12.2	Function Documentation . . . . .	175
11.12.2.1	parenthesis_structure . . . . .	175

---

11.12.2.2	parenthesis_zuker	175
11.12.2.3	energy_of_move	175
11.12.2.4	energy_of_move_pt	176
11.12.2.5	loop_energy	176
11.12.2.6	assign_plist_from_db	177
11.12.2.7	LoopEnergy	177
11.12.2.8	HairpinE	177
11.12.2.9	initialize_fold	177
11.12.2.10	energy_of_struct	178
11.12.2.11	energy_of_struct_pt	178
11.12.2.12	energy_of_circ_struct	179
11.13	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold_ vars.h File Reference	180
11.13.1	Detailed Description	182
11.13.2	Function Documentation	182
11.13.2.1	set_model_details	182
11.13.3	Variable Documentation	182
11.13.3.1	noLonelyPairs	182
11.13.3.2	dangles	182
11.13.3.3	tetra_loop	183
11.13.3.4	energy_set	183
11.13.3.5	oldAliEn	183
11.13.3.6	ribo	183
11.13.3.7	RibosumFile	183
11.13.3.8	nonstandards	183
11.13.3.9	temperature	183
11.13.3.10	ames_rule	183
11.13.3.11	llogML	183
11.13.3.12	cut_point	184
11.13.3.13	base_pair	184
11.13.3.14	pr	184
11.13.3.15	indx	184
11.13.3.16	pf_scale	184
11.13.3.17	do_backtrack	185

11.13.3.18backtrack_type . . . . .	185
11.14/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/gquad.h	
File Reference . . . . .	186
11.14.1 Detailed Description . . . . .	186
11.14.2 Function Documentation . . . . .	186
11.14.2.1 get_gquad_matrix . . . . .	186
11.14.2.2 parse_gquad . . . . .	187
11.14.2.3 backtrack_GQuad_IntLoop . . . . .	187
11.14.2.4 backtrack_GQuad_IntLoop_L . . . . .	187
11.15/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/inverse.h	
File Reference . . . . .	189
11.15.1 Detailed Description . . . . .	189
11.16/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Lfold.h	
File Reference . . . . .	190
11.16.1 Detailed Description . . . . .	190
11.17/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop_energies.h	
File Reference . . . . .	191
11.17.1 Detailed Description . . . . .	191
11.17.2 Function Documentation . . . . .	192
11.17.2.1 E_IntLoop . . . . .	192
11.17.2.2 E_Hairpin . . . . .	193
11.17.2.3 E_Stem . . . . .	194
11.17.2.4 exp_E_Stem . . . . .	195
11.17.2.5 exp_E_Hairpin . . . . .	196
11.17.2.6 exp_E_IntLoop . . . . .	196
11.18/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/LPfold.h	
File Reference . . . . .	198
11.18.1 Detailed Description . . . . .	198
11.18.2 Function Documentation . . . . .	199
11.18.2.1 init_pf_foldLP . . . . .	199
11.19/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/MEA.h	
File Reference . . . . .	200
11.19.1 Detailed Description . . . . .	200
11.19.2 Function Documentation . . . . .	200
11.19.2.1 MEA . . . . .	200

---

11.20/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/mm.h	
File Reference . . . . .	201
11.20.1 Detailed Description . . . . .	201
11.21/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/naview.h	
File Reference . . . . .	202
11.21.1 Detailed Description . . . . .	202
11.22/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/params.h	
File Reference . . . . .	203
11.22.1 Detailed Description . . . . .	204
11.23/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part_	
func.h File Reference . . . . .	205
11.23.1 Detailed Description . . . . .	207
11.23.2 Function Documentation . . . . .	207
11.23.2.1 init_pf_fold . . . . .	207
11.23.2.2 centroid . . . . .	207
11.23.2.3 mean_bp_dist . . . . .	207
11.23.2.4 expLoopEnergy . . . . .	207
11.23.2.5 expHairpinEnergy . . . . .	207
11.24/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part_	
func_co.h File Reference . . . . .	208
11.24.1 Detailed Description . . . . .	209
11.24.2 Function Documentation . . . . .	209
11.24.2.1 get_plist . . . . .	209
11.24.2.2 init_co_pf_fold . . . . .	210
11.25/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part_	
func_up.h File Reference . . . . .	211
11.25.1 Detailed Description . . . . .	211
11.26/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/plot_	
layouts.h File Reference . . . . .	212
11.26.1 Detailed Description . . . . .	213
11.26.2 Define Documentation . . . . .	213
11.26.2.1 VRNA_PLOT_TYPE_SIMPLE . . . . .	213
11.26.2.2 VRNA_PLOT_TYPE_NAVIEW . . . . .	213
11.26.2.3 VRNA_PLOT_TYPE_CIRCULAR . . . . .	213
11.26.3 Function Documentation . . . . .	214
11.26.3.1 simple_xy_coordinates . . . . .	214

11.26.3.2 simple_circplot_coordinates . . . . .	214
11.26.4 Variable Documentation . . . . .	215
11.26.4.1 rna_plot_type . . . . .	215
11.27/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/profiledist.h File Reference . . . . .	216
11.27.1 Detailed Description . . . . .	216
11.27.2 Function Documentation . . . . .	216
11.27.2.1 profile_edit_distance . . . . .	216
11.27.2.2 Make_bp_profile_bppm . . . . .	217
11.27.2.3 free_profile . . . . .	217
11.27.2.4 Make_bp_profile . . . . .	217
11.28/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/PS_- dot.h File Reference . . . . .	218
11.28.1 Detailed Description . . . . .	219
11.28.2 Function Documentation . . . . .	219
11.28.2.1 PS_rna_plot . . . . .	219
11.28.2.2 PS_rna_plot_a . . . . .	219
11.28.2.3 gmlRNA . . . . .	220
11.28.2.4 ssv_rna_plot . . . . .	220
11.28.2.5 svg_rna_plot . . . . .	220
11.28.2.6 xrna_plot . . . . .	221
11.28.2.7 PS_dot_plot_list . . . . .	221
11.28.2.8 aliPS_color_aln . . . . .	222
11.28.2.9 PS_dot_plot . . . . .	222
11.29/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/read_- epars.h File Reference . . . . .	223
11.29.1 Detailed Description . . . . .	223
11.30/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/RNAstruct.h File Reference . . . . .	224
11.30.1 Detailed Description . . . . .	225
11.30.2 Function Documentation . . . . .	225
11.30.2.1 b2HIT . . . . .	225
11.30.2.2 b2C . . . . .	226
11.30.2.3 b2Shapiro . . . . .	226
11.30.2.4 add_root . . . . .	226



---

11.30.2.5	<a href="#">expand_Shapiro</a>	226
11.30.2.6	<a href="#">expand_Full</a>	227
11.30.2.7	<a href="#">unexpand_Full</a>	227
11.30.2.8	<a href="#">unweight</a>	227
11.30.2.9	<a href="#">unexpand_aligned_F</a>	227
11.30.2.10	<a href="#">parse_structure</a>	228
11.31	<a href="#">/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/stringdist.h</a>	
	File Reference	229
11.31.1	Detailed Description	229
11.31.2	Function Documentation	229
11.31.2.1	<a href="#">Make_swString</a>	229
11.31.2.2	<a href="#">string_edit_distance</a>	229
11.32	<a href="#">/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/subopt.h</a>	
	File Reference	231
11.32.1	Detailed Description	231
11.33	<a href="#">/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/treedist.h</a>	
	File Reference	232
11.33.1	Detailed Description	232
11.33.2	Function Documentation	232
11.33.2.1	<a href="#">make_tree</a>	232
11.33.2.2	<a href="#">tree_edit_distance</a>	233
11.33.2.3	<a href="#">free_tree</a>	233
11.34	<a href="#">/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utills.h</a>	
	File Reference	234
11.34.1	Detailed Description	237
11.34.2	Define Documentation	237
11.34.2.1	<a href="#">VRNA_INPUT_ERROR</a>	237
11.34.2.2	<a href="#">VRNA_INPUT_QUIT</a>	237
11.34.2.3	<a href="#">VRNA_INPUT_MISC</a>	237
11.34.2.4	<a href="#">VRNA_INPUT_FASTA_HEADER</a>	237
11.34.2.5	<a href="#">VRNA_INPUT_SEQUENCE</a>	237
11.34.2.6	<a href="#">VRNA_INPUT_CONSTRAINT</a>	237
11.34.2.7	<a href="#">VRNA_INPUT_NO_TRUNCATION</a>	238
11.34.2.8	<a href="#">VRNA_INPUT_NO_REST</a>	238
11.34.2.9	<a href="#">VRNA_INPUT_NO_SPAN</a>	238

11.34.2.10	VRNA_INPUT_NOSKIP_BLANK_LINES . . . . .	238
11.34.2.11	VRNA_INPUT_BLANK_LINE . . . . .	238
11.34.2.12	VRNA_INPUT_NOSKIP_COMMENTS . . . . .	238
11.34.2.13	VRNA_INPUT_COMMENT . . . . .	238
11.34.2.14	VRNA_CONSTRAINT_PIPE . . . . .	238
11.34.2.15	VRNA_CONSTRAINT_DOT . . . . .	238
11.34.2.16	VRNA_CONSTRAINT_X . . . . .	238
11.34.2.17	VRNA_CONSTRAINT_ANG_BRACK . . . . .	239
11.34.2.18	VRNA_CONSTRAINT_RND_BRACK . . . . .	239
11.34.2.19	VRNA_CONSTRAINT_MULTILINE . . . . .	239
11.34.2.20	VRNA_CONSTRAINT_NO_HEADER . . . . .	239
11.34.2.21	VRNA_CONSTRAINT_ALL . . . . .	239
11.34.2.22	VRNA_CONSTRAINT_G . . . . .	239
11.34.2.23	VRNA_OPTION_MULTILINE . . . . .	239
11.34.2.24	MIN2 . . . . .	239
11.34.2.25	MAX2 . . . . .	239
11.34.2.26	MIN3 . . . . .	240
11.34.2.27	MAX3 . . . . .	240
11.34.2.28	XSTR . . . . .	240
11.34.2.29	STR . . . . .	240
11.34.2.30	FILENAME_MAX_LENGTH . . . . .	240
11.34.2.31	FILENAME_ID_LENGTH . . . . .	240
11.34.3	Function Documentation . . . . .	240
11.34.3.1	space . . . . .	240
11.34.3.2	xrealloc . . . . .	241
11.34.3.3	nerror . . . . .	241
11.34.3.4	warn_user . . . . .	241
11.34.3.5	urn . . . . .	241
11.34.3.6	int_urn . . . . .	242
11.34.3.7	time_stamp . . . . .	242
11.34.3.8	random_string . . . . .	242
11.34.3.9	hamming . . . . .	242
11.34.3.10	hamming_bound . . . . .	243
11.34.3.11	lget_line . . . . .	243

---

11.34.3.12	get_input_line	243
11.34.3.13	read_record	244
11.34.3.14	pack_structure	245
11.34.3.15	unpack_structure	245
11.34.3.16	make_pair_table	246
11.34.3.17	copy_pair_table	246
11.34.3.18	make_pair_table	246
11.34.3.19	make_pair_table_snoop	246
11.34.3.20	make_loop_index_pt	246
11.34.3.21	print_tty_input_seq	247
11.34.3.22	print_tty_input_seq_str	247
11.34.3.23	print_tty_constraint	247
11.34.3.24	str_DNA2RNA	247
11.34.3.25	str_uppercase	248
11.34.3.26	get_iindx	248
11.34.3.27	get_indx	248
11.34.3.28	constrain_ptypes	249
11.34.4	Variable Documentation	249
11.34.4.1	xsubi	249
11.35	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/lib/1.8.4_ - epars.h File Reference	250
11.35.1	Detailed Description	250
11.36	/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/lib/1.8.4_ - intloops.h File Reference	251
11.36.1	Detailed Description	251



# Chapter 1

## ViennaRNA Package core - RNAlib

**A Library for folding and comparing RNA secondary structures**

**Date:**

1994-2012

**Authors:**

Ivo Hofacker, Peter Stadler, Ronny Lorenz and many more

### Table of Contents

- [Introduction](#)
- [RNA Secondary Structure Folding](#)
- [Parsing and Comparing - Functions to Manipulate Structures](#)
- [Utilities - Odds and Ends](#)
- [Example - A Small Example Program](#)
- [mp\\_ref](#)

## 1.1 Introduction

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

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

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

## **Chapter 2**

# **Parsing and Comparing - Functions to Manipulate Structures**

## Representations of Secondary Structures

The standard representation of a secondary structure is the *bracket notation*, where matching brackets symbolize base pairs and unpaired bases are shown as dots. Alternatively, one may use two types of node labels, 'P' for paired and 'U' for unpaired; a dot is then replaced by '(U)', and each closed bracket is assigned an additional identifier 'P'. We call this the expanded notation. In fontana:1993b a condensed representation of the secondary structure is proposed, the so-called homeomorphically irreducible tree (HIT) representation. Here a stack is represented as a single pair of matching brackets labeled 'P' and weighted by the number of base pairs. Correspondingly, a contiguous strain of unpaired bases is shown as one pair of matching brackets labeled 'U' and weighted by its length. Generally any string consisting of matching brackets and identifiers is equivalent to a plane tree with as many different types of nodes as there are identifiers.

Bruce Shapiro proposed a coarse grained representation shapiro:1988, which, does not retain the full information of the secondary structure. He represents the different structure elements by single matching brackets and labels them as 'H' (hairpin loop), 'I' (interior loop), 'B' (bulge), 'M' (multi-loop), and 'S' (stack). We extend his alphabet by an extra letter for external elements 'E'. Again these identifiers may be followed by a weight corresponding to the number of unpaired bases or base pairs in the structure element. All tree representations (except for the dot-bracket form) can be encapsulated into a virtual root (labeled 'R'), see the example below.

The following example illustrates the different linear tree representations used by the package. All lines show the same secondary structure.

```
a) . (((((...(((...)))...((...)))...)).
    (U) (((((U) (U) (((U) (U) (U) P) P) P) (U) (U) ((U) (U) P) P) P) (U) P) P) (U)
b) (U) (((U2) ((U3) P3) (U2) ((U2) P2) P2) (U) P2) (U)
c) ((H) (H) M) B)
    (((((H) S) ((H) S) M) S) B) S)
    ((((((H) S) ((H) S) M) S) B) S) E)
d) (((((((H3) S3) ((H2) S2) M4) S2) B1) S2) E2) R)
```

Above: [Tree](#) representations of secondary structures. a) Full structure: the first line shows the more convenient condensed notation which is used by our programs; the second line shows the rather clumsy expanded notation for completeness, b) HIT structure, c) different versions of coarse grained structures: the second line is exactly Shapiro's representation, the first line is obtained by neglecting the stems. Since each loop is closed by a unique stem, these two lines are equivalent. The third line is an extension taking into account also the external digits. d) weighted coarse structure, this time including the virtual root.

For the output of aligned structures from string editing, different representations are needed, where we put the label on both sides. The above examples for tree representations would then look like:

```
a) (UU) (P (P (P (P (UU) (UU) (P (P (P (UU) (UU) (UU) P) P) P) (UU) (UU) (P (P (UU) (U...
b) (UU) (P2 (P2 (U2U2) (P2 (U3U3) P3) (U2U2) (P2 (U2U2) P2) P2) (UU) P2) (UU)
c) (B (M (HH) (HH) M) B)
    (S (B (S (M (S (HH) S) (S (HH) S) M) S) B) S)
    (E (S (B (S (M (S (HH) S) (S (HH) S) M) S) B) S) E)
d) (R (E2 (S2 (B1 (S2 (M4 (S3 (H3) S3) ((H2) S2) M4) S2) B1) S2) E2) R)
```



---

Aligned structures additionally contain the gap character '\_'.

## Parsing and Coarse Graining of Structures

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

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

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

## 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)
```

For other cases a distance measure that allows for gaps is preferable. We can define distances between structures as edit distances between trees or their string representations. In the case of string distances this is the same as "sequence alignment". Given a set of edit operations and edit costs, the edit distance is given by the minimum sum of the costs along an edit path converting one object into the other. Edit distances like these always define a metric. The edit operations used by us are insertion, deletion and replacement of nodes. String editing does not pay attention to the matching of brackets, while in tree editing matching brackets represent a single node of the tree. [Tree editing](#) is therefore usually preferable, although somewhat slower. String edit distances are always smaller or equal to tree edit distances.

The different level of detail in the structure representations defined above naturally leads to different measures of distance. For full structures we use a cost of 1 for deletion or insertion of an unpaired base and 2 for a base pair. Replacing an unpaired base for a pair incurs a cost of 1.

Two cost matrices are provided for coarse grained structures:

```
/* Null, H, B, I, M, S, E */
{ 0, 2, 2, 2, 2, 1, 1}, /* Null replaced */
{ 2, 0, 2, 2, 2, INF, INF}, /* H replaced */
{ 2, 2, 0, 1, 2, INF, INF}, /* B replaced */
{ 2, 2, 1, 0, 2, INF, INF}, /* I replaced */
{ 2, 2, 2, 2, 0, INF, INF}, /* M replaced */
{ 1, INF, INF, INF, INF, 0, INF}, /* S replaced */
{ 1, INF, INF, INF, INF, INF, 0}, /* E replaced */

/* Null, H, B, I, M, S, E */
{ 0, 100, 5, 5, 75, 5, 5}, /* Null replaced */
{ 100, 0, 8, 8, 8, INF, INF}, /* H replaced */
{ 5, 8, 0, 3, 8, INF, INF}, /* B replaced */
```

---

```

{ 5, 8, 3, 0, 8, INF, INF}, /* I replaced */
{ 75, 8, 8, 8, 0, INF, INF}, /* M replaced */
{ 5, INF, INF, INF, INF, 0, INF}, /* S replaced */
{ 5, INF, INF, INF, INF, INF, 0}, /* E replaced */

```

The lower matrix uses the costs given in shapiro:1990. All distance functions use the following global variables:

```
int cost_matrix;
```

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

**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

### 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

### Functions for Comparison of Base Pair Probabilities

For comparison of base pair probability matrices, the matrices are first condensed into probability profiles which are then 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

[Next Page: Utilities](#)

## **Chapter 3**

# **Utilities - Odds and Ends**

**Table of Contents**

- [Producing secondary structure graphs](#)
- [Producing \(colored\) dot plots for base pair probabilities](#)
- [Producing \(colored\) alignments](#)
- [RNA sequence related utilities](#)
- [RNA secondary structure related utilities](#)
- [Miscellaneous Utilities](#)

**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 layouting 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.

## 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.

### 3.3 Producing (colored) alignments

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

### 3.4 RNA sequence related utilities

Several functions provide useful applications to RNA sequences

```
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.

```
void str_DNA2RNA(char *sequence);
```

Convert a DNA input sequence to RNA alphabet.

```
void str_uppercase(char *sequence);
```

Convert an input sequence to uppercase.

### 3.5 RNA secondary structure related utilities

```
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.



## 3.6 Miscellaneous Utilities

```
void print_tty_input_seq (void)
```

Print a line to *stdout* that asks for an input sequence.

```
void print_tty_constraint_full (void)
```

Print structure constraint characters to *stdout* (full constraint support).

```
void print_tty_constraint (unsigned int option)
```

Print structure constraint characters to *stdout*. (constraint support is specified by option parameter).

```
int *get_iindx (unsigned int length)
```

Get an index mapper array (*iindx*) for accessing the energy matrices, e.g. in partition function related functions.

```
int *get_indx (unsigned int length)
```

Get an index mapper array (*indx*) for accessing the energy matrices, e.g. in MFE related functions.

```
void constrain_ptypes (
    const char *constraint,
    unsigned int length,
    char *ptype,
    int *BP,
    int min_loop_size,
    unsigned int idx_type)
```

Insert constraining pair types according to constraint structure string.

```
char *get_line(FILE *fp);
```

Read a line of arbitrary length from a stream.

```
unsigned int read_record(
    char **header,
    char **sequence,
    char ***rest,
    unsigned int options);
```

Get a data record from *stdin*.

```
char *time_stamp (void)
```

Get a timestamp.

```
void warn_user (const char message[])
```

Print a warning message.

```
void nrerror (const char message[])
```

Die with an error message.

```
void init_rand (void)
```

Make random number seeds.

```
unsigned short xsubi[3];
```

Current 48 bit random number.

```
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 *space (unsigned size)
```

Allocate space safely.

```
void *xrealloc ( void *p,  
                unsigned size)
```

Reallocate space safely.

**See also:**

[utils.h](#) for a complete overview and detailed description of the utility functions

[Next Page: Examples](#)

## **Chapter 4**

### **Example - A Small Example Program**

The following program exercises most commonly used functions of the library. The program folds two sequences using both the mfe and partition function algorithms and calculates the tree edit and profile distance of the resulting structures and base pairing probabilities.

```

{.c}
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "utils.h"
#include "fold_vars.h"
#include "fold.h"
#include "part_func.h"
#include "inverse.h"
#include "RNAstruct.h"
#include "treedist.h"
#include "stringdist.h"
#include "profiledist.h"

void main()
{
    char *seq1="CGCAGGGAUACCCGCG", *seq2="GCGCCCAUAGGGACGC",
        *struct1,* struct2,* xstruc;
    float e1, e2, tree_dist, string_dist, profile_dist, kT;
    Tree *T1, *T2;
    swString *S1, *S2;
    float *pf1, *pf2;
    FLT_OR_DBL *bppm;
    /* fold at 30C instead of the default 37C */
    temperature = 30.; /* must be set *before* initializing */

    /* allocate memory for structure and fold */
    struct1 = (char* ) space(sizeof(char)*(strlen(seq1)+1));
    e1 = fold(seq1, struct1);

    struct2 = (char* ) space(sizeof(char)*(strlen(seq2)+1));
    e2 = fold(seq2, struct2);

    free_arrays(); /* free arrays used in fold() */

    /* produce tree and string representations for comparison */
    xstruc = expand_Full(struct1);
    T1 = make_tree(xstruc);
    S1 = Make_swString(xstruc);
    free(xstruc);

    xstruc = expand_Full(struct2);
    T2 = make_tree(xstruc);
    S2 = Make_swString(xstruc);
    free(xstruc);

    /* calculate tree edit distance and aligned structures with gaps */
    edit_backtrack = 1;
    tree_dist = tree_edit_distance(T1, T2);
    free_tree(T1); free_tree(T2);
    unexpand_aligned_F(aligned_line);
    printf("%s\n%s %3.2f\n", aligned_line[0], aligned_line[1], tree_dist);

    /* same thing using string edit (alignment) distance */
    string_dist = string_edit_distance(S1, S2);

```

---

```

free(S1); free(S2);
printf("%s mfe=%5.2f\n%s mfe=%5.2f dist=%3.2f\n",
       aligned_line[0], e1, aligned_line[1], e2, string_dist);

/* for longer sequences one should also set a scaling factor for
   partition function folding, e.g: */
kT = (temperature+273.15)*1.98717/1000.; /* kT in kcal/mol */
pf_scale = exp(-e1/kT/strlen(seq1));

/* calculate partition function and base pair probabilities */
e1 = pf_fold(seq1, struct1);
/* get the base pair probability matrix for the previous run of pf_fold() */
bppm = export_bppm();
pf1 = Make_bp_profile_bppm(bppm, strlen(seq1));

e2 = pf_fold(seq2, struct2);
/* get the base pair probability matrix for the previous run of pf_fold() */
bppm = export_bppm();
pf2 = Make_bp_profile_bppm(bppm, strlen(seq2));

free_pf_arrays(); /* free space allocated for pf_fold() */

profile_dist = profile_edit_distance(pf1, pf2);
printf("%s free energy=%5.2f\n%s free energy=%5.2f dist=%3.2f\n",
       aligned_line[0], e1, aligned_line[1], e2, profile_dist);

free_profile(pf1); free_profile(pf2);
}

```

In a typical Unix environment you would compile this program using:

```
cc ${OPENMP_CFLAGS} -c example.c -I${hpath}
```

and link using

```
cc ${OPENMP_CFLAGS} -o example -L${lpath} -lRNA -lm
```

where *hpath* and *lpath* point to the location of the header files and library, respectively.

**Note:**

As default, the RNAlib is compiled with build-in *OpenMP* multithreading support. Thus, when linking your own object files to the library you have to pass the compiler specific *OPENMP\_CFLAGS* (e.g. '-fopenmp' for **gcc**) even if your code does not use openmp specific code. However, in that case the *OpenMP* flags may be omitted when compiling example.c



## **Chapter 5**

### **Deprecated List**

- 
- Global `base_pair`** Do not use this variable anymore!
- Global `centroid`** This function is deprecated and should not be used anymore as it is not threadsafe!
- Global `energy_of_circ_struct`** This function is deprecated and should not be used in future programs Use `energy_of_circ_structure()` instead!
- Global `energy_of_struct`** This function is deprecated and should not be used in future programs! Use `energy_of_structure()` instead!
- Global `energy_of_struct_pt`** This function is deprecated and should not be used in future programs! Use `energy_of_structure_pt()` instead!
- Global `expHairpinEnergy`** Use `exp_E_Hairpin()` from `loop_energies.h` instead
- Global `expLoopEnergy`** Use `exp_E_IntLoop()` from `loop_energies.h` instead
- Global `get_plist`** { This function is deprecated and will be removed soon!} use `assign_plist_from_pr()` instead!
- Global `HairpinE`** {This function is deprecated and will be removed soon. Use `E_-Hairpin()` instead!}
- Global `iindx`** Do not use this variable anymore!
- Global `init_co_pf_fold`** { This function is deprecated and will be removed soon!}
- Global `init_pf_fold`** This function is obsolete and will be removed soon!
- Global `initialize_cofold`** {This function is obsolete and will be removed soon!}
- Global `initialize_fold`** {This function is deprecated and will be removed soon!}
- Global `LoopEnergy`** {This function is deprecated and will be removed soon. Use `E_IntLoop()` instead!}



**Global [Make\\_bp\\_profile](#)** This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement

**Global [mean\\_bp\\_dist](#)** This function is not threadsafe and should not be used anymore. Use [mean\\_bp\\_distance\(\)](#) instead!

**Global [pr](#)** Do not use this variable anymore!

**Global [PS\\_dot\\_plot](#)** This function is deprecated and will be removed soon! Use [PS\\_dot\\_plot\\_list\(\)](#) instead!



# Chapter 6

## Module Index

### 6.1 Modules

Here is a list of all modules:

RNA Secondary Structure Folding . . . . .	31
Calculating Minimum Free Energy (MFE) Structures . . . . .	33
MFE Structures of two hybridized Sequences . . . . .	55
MFE Consensus Structures for Sequence Alignment(s) . . . . .	71
Local MFE structure Prediction and Z-scores . . . . .	77
Calculating MFE representatives of a Distance Based Partitioning . . . . .	101
Calculating Partition Functions and Pair Probabilities . . . . .	37
Compute the structure with maximum expected accuracy (MEA) . . . . .	45
Compute the centroid structure . . . . .	46
Partition Function for two hybridized Sequences . . . . .	58
Partition Function for two hybridized Sequences as a stepwise Process . . . . .	63
Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . . .	73
Partition functions for locally stable secondary structures . . . . .	78
Calculate Partition Functions of a Distance Based Partitioning . . . . .	104
Enumerating Suboptimal Structures . . . . .	48
Suboptimal structures according to Zuker et al. 1989 . . . . .	49
Suboptimal structures within an energy band around the MFE . . . . .	50
Stochastic backtracking in the Ensemble . . . . .	52
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) . . . . .	75
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	107
Calculate Secondary Structures of two RNAs upon Dimerization . . . . .	54
MFE Structures of two hybridized Sequences . . . . .	55
Partition Function for two hybridized Sequences . . . . .	58
Partition Function for two hybridized Sequences as a stepwise Process . . . . .	63
Predicting Consensus Structures from Alignment(s) . . . . .	66
MFE Consensus Structures for Sequence Alignment(s) . . . . .	71

---

Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . . .	73
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) . . . . .	75
Local MFE consensus structures for Sequence Alignments . . . . .	81
Predicting Locally stable structures of large sequences . . . . .	76
Local MFE structure Prediction and Z-scores . . . . .	77
Partition functions for locally stable secondary structures . . . . .	78
Local MFE consensus structures for Sequence Alignments . . . . .	81
Change and Precalculate Energy Parameter Sets and Boltzmann Factors . . . . .	82
Reading/Writing energy parameter sets from/to File . . . . .	86
Converting energy parameter files . . . . .	88
Energy evaluation . . . . .	93
Searching Sequences for Predefined Structures . . . . .	97
Classified Dynamic Programming . . . . .	99
Distance based partitioning of the Secondary Structure Space . . . . .	100
Calculating MFE representatives of a Distance Based Partitioning . . . . .	101
Calculate Partition Functions of a Distance Based Partitioning . . . . .	104
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	107
Compute the Density of States . . . . .	109
Parsing and Comparing - Functions to Manipulate Structures . . . . .	110

# Chapter 7

## Data Structure Index

### 7.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">bondT</a> (Base pair ) . . . . .	111
<a href="#">bondTEn</a> (Base pair with associated energy ) . . . . .	112
<a href="#">cofoldF</a> . . . . .	113
<a href="#">ConcEnt</a> . . . . .	114
<a href="#">constrain</a> (Constraints for cofolding ) . . . . .	115
<a href="#">COORDINATE</a> (This is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type <a href="#">COORDINATE</a> ) . . . . .	116
<a href="#">cpair</a> (This datastructure is used as input parameter in functions of <a href="#">PS_dot.c</a> )	117
<a href="#">duplexT</a> . . . . .	118
<a href="#">dupVar</a> . . . . .	119
<a href="#">folden</a> . . . . .	120
<a href="#">interact</a> . . . . .	121
<a href="#">intermediate_t</a> . . . . .	122
<a href="#">INTERVAL</a> (Sequence interval stack element used in <a href="#">subopt.c</a> ) . . . . .	123
<a href="#">LIST</a> . . . . .	124
<a href="#">LST_BUCKET</a> . . . . .	125
<a href="#">model_detailsT</a> (The data structure that contains the complete model details used throughout the calculations ) . . . . .	126
<a href="#">move_t</a> . . . . .	128
<a href="#">PAIR</a> (Base pair data structure used in <a href="#">subopt.c</a> ) . . . . .	129
<a href="#">pair_info</a> (A base pair info structure ) . . . . .	130
<a href="#">pairpro</a> . . . . .	131
<a href="#">paramT</a> (The datastructure that contains temperature scaled energy param- eters ) . . . . .	132
<a href="#">path_t</a> . . . . .	133
<a href="#">pf_paramT</a> (The datastructure that contains temperature scaled Boltzmann weights of the energy parameters ) . . . . .	134
<a href="#">plist</a> (This datastructure is used as input parameter in functions of <a href="#">PS_dot.h</a> and others ) . . . . .	136

---

Postorder_list	137
pu_contrib (Contributions to p_u)	138
pu_out (Collection of all free_energy of being unpaired values for output)	139
sect (Stack of partial structures for backtracking)	140
snoopT	141
SOLUTION (Solution element from subopt.c)	142
struct_en	143
svm_model	144
swString	145
Tree	146
TwoDfold_solution (Solution element returned from TwoDfoldList)	147
TwoDfold_vars (Variables compound for 2Dfold MFE folding)	148
TwoDpfold_solution (Solution element returned from TwoDpfoldList)	150
TwoDpfold_vars (Variables compound for 2Dfold partition function folding)	151

# Chapter 8

## File Index

### 8.1 File List

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

<b>mainpage.h</b> . . . . .	??
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/2Dfold.h . . . . .	153
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/2Dpfold.h . . . . .	155
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/ali_plex.h . . . . .	??
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/alifold.h (Compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments) . . . . .	156
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/aln_util.h . . . . .	??
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/cofold.h (MFE version of cofolding routines) . . . . .	159
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/convert_epars.h (Functions and definitions for energy parameter file format conversion) . . . . .	161
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h (All datastructures and typedefs shared among the Vienna RNA Package can be found here) . . . . .	162
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist_vars.h (Global variables for Distance-Package) . . . . .	165
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/duplex.h (Duplex folding function declarations..) . . . . .	167
/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/edit_cost.h (Global variables for Edit Costs included by treedist.c and stringdist.c) . . . . .	168

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/energy_const.h	169
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/energy_par.h	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/findpath.h (Compute direct refolding paths between two secondary structures)	171
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h (MFE calculations and energy evaluations for single RNA sequences)	173
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold_vars.h (Here all all declarations of the global variables used throughout RNAlib)	180
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/gquad.h (Various functions related to G-quadruplex computations)	186
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/inverse.h (Inverse folding routines)	189
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Lfold.h (Predicting local MFE structures of large sequences)	190
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop_energies.h (Energy evaluation for MFE and partition function calculations)	191
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/LPfold.h (Function declarations of partition function variants of the Lfold algorithm)	198
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/MEA.h (Computes a MEA (maximum expected accuracy) structure)	200
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/mm.h (Several Maximum Matching implementations)	201
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/move_set.h	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/naview.h	202
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/pair_mat.h	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/params.h	203
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part_func.h (Partition function of single RNA sequences)	205
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part_func_co.h (Partition function for two RNA sequences)	208
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part_func_up.h (Partition Function Cofolding as stepwise process)	211



/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>PKplex.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>plex.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>plot_layouts.h</b> (Secondary structure plot layout algorithms ) . . . . .	212
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>ProfileAln.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>profiledist.h</b> . . . . .	216
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>PS_dot.h</b> (Various functions for plotting RNA secondary structures, dot-plots and other visualizations ) . . . . .	218
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>read_epars.h</b> . . . . .	223
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>ribo.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>RNAstruct.h</b> (Parsing and Coarse Graining of Structures ) . . . . .	224
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>snofold.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>snoop.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>stringdist.h</b> (Functions for String Alignment ) . . . . .	229
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>subopt.h</b> (RNAsubopt and density of states declarations ) . . . . .	231
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/ <b>svm_utils.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>treedist.h</b> (Functions for <b>Tree</b> Edit Distances ) . . . . .	232
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/H/ <b>utils.h</b> (Various utility- and helper-functions used throughout the Vienna RNA package ) . . . . .	234
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/lib/1.8.4_ <b>epars.h</b> (Free energy parameters for parameter file conversion ) . . . . .	250
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/lib/1.8.4_ <b>intloops.h</b> (Free energy parameters for interior loop contributions needed by the parameter file conversion functions ) . . . . .	251
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/lib/ <b>intl11.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/lib/ <b>intl11dH.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR-NA/lib/ <b>intl21.h</b> . . . . .	??

---

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR- NA/lib/ <b>intl21dH.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR- NA/lib/ <b>intl22.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR- NA/lib/ <b>intl22dH.h</b> . . . . .	??
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR- NA/lib/ <b>list.h</b> . . . . .	??

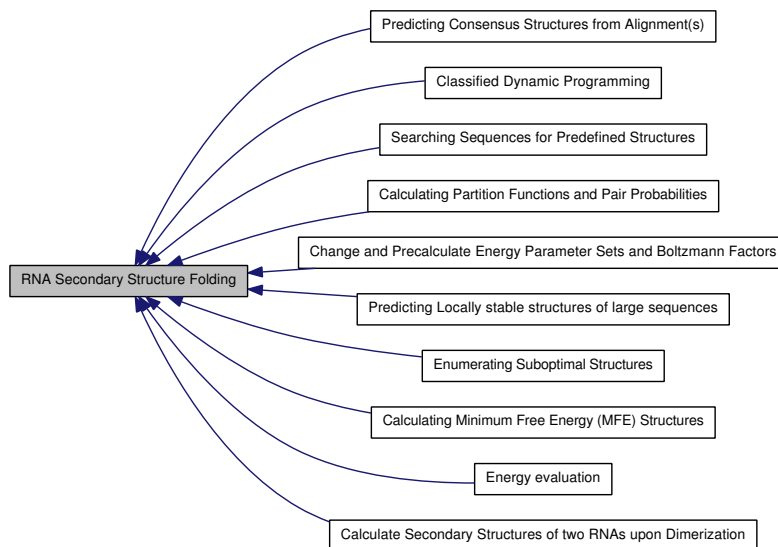
# Chapter 9

## Module Documentation

### 9.1 RNA Secondary Structure Folding

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

Collaboration diagram for RNA Secondary Structure Folding:



#### Modules

- [Calculating Minimum Free Energy \(MFE\) Structures](#)

*This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.*

- [Calculating Partition Functions and Pair Probabilities](#)

*This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.*

- [Enumerating Suboptimal Structures](#)
- [Calculate Secondary Structures of two RNAs upon Dimerization](#)

*Predict structures formed by two molecules upon hybridization.*

- [Predicting Consensus Structures from Alignment\(s\)](#)

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

- [Predicting Locally stable structures of large sequences](#)
- [Change and Precalculate Energy Parameter Sets and Boltzmann Factors](#)

*All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).*

- [Energy evaluation](#)

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

- [Searching Sequences for Predefined Structures](#)
- [Classified Dynamic Programming](#)

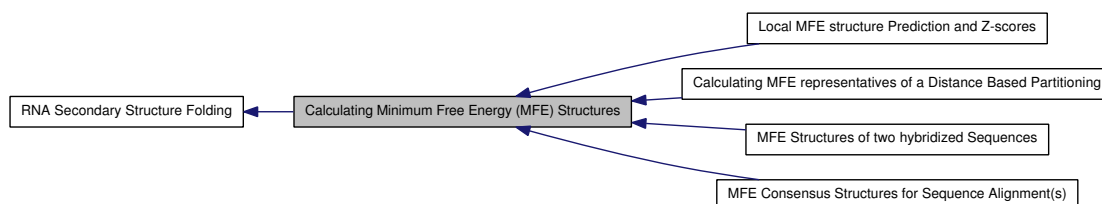
### 9.1.1 Detailed Description

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

## 9.2 Calculating Minimum Free Energy (MFE) Structures

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

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



### Modules

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

*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.*

### Functions

- float [fold\\_par](#) (const char \*sequence, char \*structure, [paramT](#) \*parameters, int is\_constrained, int is\_circular)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [fold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [circfold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.*
- void [free\\_arrays](#) (void)  
*Free arrays for mfe folding.*
- void [update\\_fold\\_params](#) (void)

*Recalculate energy parameters.*

### 9.2.1 Detailed Description

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences. This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

The library provides a fast dynamic programming minimum free energy folding algorithm as described by Zuker & Stiegler (1981).

The library provides a fast dynamic programming minimum free energy folding algorithm as described in zuker:1981. All relevant parts that directly implement the "Zuker & Stiegler" algorithm for single sequences are described in this section.

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

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

### 9.2.2 Function Documentation

#### 9.2.2.1 `float fold_par (const char * sequence, char * structure, paramT * parameters, int is_constrained, int is_circular)`

Compute minimum free energy and an appropriate secondary structure of an RNA sequence. The first parameter given, the RNA sequence, must be *uppercase* and should only contain an alphabet  $\Sigma$  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 `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

**See also:**

[fold\(\)](#), [circfold\(\)](#), [model\\_detailsT](#), [set\\_energy\\_model\(\)](#), [get\\_scaled\\_parameters\(\)](#)

**Parameters:**

*sequence* RNA sequence

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

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

*is\_constrained* Switch to indicate that a structure constraint is passed via the structure argument (0==off)

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

**Returns:**

the minimum free energy (MFE) in kcal/mol

**9.2.2.2 float fold (const char \* sequence, char \* structure)**

Compute minimum free energy and an appropriate secondary structure of an RNA sequence. This function essentially does the same thing as [fold\\_par\(\)](#). However, it takes its model details, i.e. [temperature](#), [dangles](#), [tetra\\_loop](#), [noGU](#), [no\\_closingGU](#), [fold\\_constrained](#), [noLonelyPairs](#) from the current global settings within the library

Use [fold\\_par\(\)](#) for a completely threadsafe variant

**See also:**

[fold\\_par\(\)](#), [circfold\(\)](#)

**Parameters:**

*sequence* RNA sequence

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

**Returns:**

the minimum free energy (MFE) in kcal/mol

### 9.2.2.3 float circfold (const char \* *sequence*, char \* *structure*)

Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence. This function essentially does the same thing as [fold\\_par\(\)](#). However, it takes its model details, i.e. [temperature](#), [dangles](#), [tetra\\_loop](#), [noGU](#), [no\\_closingGU](#), [fold\\_constrained](#), [noLonelyPairs](#) from the current global settings within the library

Use [fold\\_par\(\)](#) for a completely threadsafe variant

**See also:**

[fold\\_par\(\)](#), [circfold\(\)](#)

**Parameters:**

*sequence* RNA sequence

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

**Returns:**

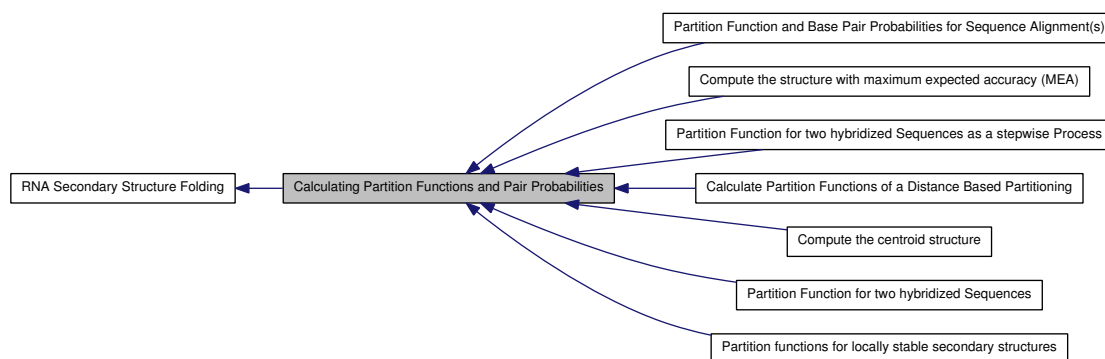
the minimum free energy (MFE) in kcal/mol



## 9.3 Calculating Partition Functions and Pair Probabilities

This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.

Collaboration diagram for Calculating Partition Functions and Pair Probabilities:



### Modules

- [Compute the structure with maximum expected accuracy \(MEA\)](#)
- [Compute the centroid structure](#)
- [Partition Function for two hybridized Sequences](#)

*Partition Function Cofolding.*

- [Partition Function for two hybridized Sequences as a stepwise Process](#)

*Partition Function Cofolding as a stepwise process.*

- [Partition Function and Base Pair Probabilities for Sequence Alignment\(s\)](#)
- [Partition functions for locally stable secondary structures](#)
- [Calculate Partition Functions of a Distance Based Partitioning](#)

*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.*

### Functions

- float [pf\\_fold\\_par](#) (const char \*sequence, char \*structure, [pf\\_paramT](#) \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)

*Compute the partition function  $Q$  for a given RNA sequence.*

- float [pf\\_fold](#) (const char \*sequence, char \*structure)

*Compute the partition function  $Q$  of an RNA sequence.*

- float `pf_circ_fold` (const char \*sequence, char \*structure)

*Compute the partition function of a circular RNA sequence.*

- void `free_pf_arrays` (void)

*Free arrays for the partition function recursions.*

- void `update_pf_params` (int length)

*Recalculate energy parameters.*

- void `update_pf_params_par` (int length, `pf_paramT` \*parameters)

*Recalculate energy parameters.*

- double \* `export_bppm` (void)

*Get a pointer to the base pair probability array*

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

- void `assign_plist_from_pr` (`plist` \*\*pl, double \*probs, int length, double cutoff)

*Create a `plist` from a probability matrix.*

- int `get_pf_arrays` (short \*\*S\_p, short \*\*S1\_p, char \*\*ptype\_p, double \*\*qb\_p, double \*\*qm\_p, double \*\*q1k\_p, double \*\*qln\_p)

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

- double `mean_bp_distance` (int length)

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

- double `mean_bp_distance_pr` (int length, double \*pr)

*Get the mean base pair distance in the thermodynamic ensemble.*

### 9.3.1 Detailed Description

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

### 9.3.2 Function Documentation

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

Compute the partition function  $Q$  for a given RNA sequence. If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters ". , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If *fold\_constrained* is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If 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:

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

#### Postcondition:

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

#### See also:

`pf_fold()`, `pf_circ_fold()`, `bppm_to_structure()`, `export_bppm()`, `get_boltzmann_factors()`, `free_pf_arrays()`

#### Parameters:

- ← *sequence* The RNA sequence input
- ↔ *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)
- ← *parameters* Data structure containing the precalculated Boltzmann factors
- ← *calculate\_bppm* Switch to Base pair probability calculations on/off (0==off)
- ← *is\_constrained* Switch to indicate that a structure constraint is passed via the structure argument (0==off)
- ← *is\_circular* Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

#### Returns:

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

### 9.3.2.2 float pf\_fold (const char \* sequence, char \* structure)

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

#### Note:

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

**OpenMP:** This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using `pf_fold_par()` for a really threadsafe implementation.

#### Precondition:

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

#### Postcondition:

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

#### See also:

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

#### Parameters:

*sequence* The RNA sequence input

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

#### Returns:

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

### 9.3.2.3 float pf\_circ\_fold (const char \* sequence, char \* structure)

Compute the partition function of a circular RNA sequence.

**Note:**

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

**OpenMP:** This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using `pf_fold_par()` for a really threadsafe implementation.

**Precondition:**

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

**Postcondition:**

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

**See also:**

`pf_fold_par()`, `pf_fold()`

**Parameters:**

← *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

### 9.3.2.4 void free\_pf\_arrays (void)

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

**Note:**

Successive calls of `pf_fold()`, `pf_circ_fold()` already check if they should free any memory from a previous run.

**OpenMP notice:**

This function should be called before leaving a thread in order to avoid leaking memory

**Postcondition:**

All memory allocated by [pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#) or [pf\\_circ\\_fold\(\)](#) will be free'd

**See also:**

[pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#), [pf\\_circ\\_fold\(\)](#)

**9.3.2.5 void update\_pf\_params (int length)**

Recalculate energy parameters. Call this function to recalculate the pair matrix and energy parameters after a change in folding parameters like [temperature](#)

**9.3.2.6 double\* export\_bppm (void)**

Get a pointer to the base pair probability array

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

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

**Precondition:**

Call [pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#) or [pf\\_circ\\_fold\(\)](#) first to fill the base pair probability array

**See also:**

[pf\\_fold\(\)](#), [pf\\_circ\\_fold\(\)](#), [get\\_iindx\(\)](#)

**Returns:**

A pointer to the base pair probability array

**9.3.2.7 void assign\_plist\_from\_pr (plist \*\* pl, double \* probs, int length, double cutoff)**

Create a [plist](#) from a probability matrix. The probability matrix given is parsed and all pair probabilities above the given threshold are used to create an entry in the [plist](#)

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

**Note:**

This function is threadsafe

**Parameters:**

→ *pl* A pointer to the [plist](#) that is to be created

- ← *probs* The probability matrix used for creting the [plist](#)
- ← *length* The length of the RNA sequence
- ← *cutoff* The cutoff value

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

Get the pointers to (almost) all relevant 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:**

- *S\_p* A pointer to the 'S' array (integer representation of nucleotides)
- *S1\_p* A pointer to the 'S1' array (2nd integer representation of nucleotides)
- *ptype\_p* A pointer to the pair type matrix
- *qb\_p* A pointer to the  $Q^B$  matrix
- *qm\_p* A pointer to the  $Q^M$  matrix
- *q1k\_p* A pointer to the 5' slice of the Q matrix ( $q1k(k) = Q(1, k)$ )
- *qln\_p* A pointer to the 3' slice of the Q matrix ( $qln(l) = Q(l, n)$ )

**Returns:**

Non Zero if everything went fine, 0 otherwise

**9.3.2.9** `double mean_bp_distance (int length)`

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

**Note:**

To ensure thread-safety, use the function [mean\\_bp\\_distance\\_pr\(\)](#) instead!

**See also:**

[mean\\_bp\\_distance\\_pr\(\)](#)

**Parameters:**

*length*

**Returns:**

mean base pair distance in thermodynamic ensemble

### 9.3.2.10 double mean\_bp\_distance\_pr (int *length*, double \* *pr*)

Get the mean base pair distance in the thermodynamic ensemble. This is a threadsafe implementation of [mean\\_bp\\_dist\(\)](#) !

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

this can be computed from the pair probs  $p_{ij}$  as

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

**Note:**

This function is threadsafe

**Parameters:**

*length* The length of the sequence

*pr* The matrix containing the base pair probabilities

**Returns:**

The mean pair distance of the structure ensemble



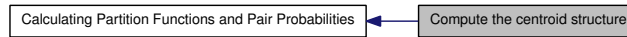
## 9.4 Compute the structure with maximum expected accuracy (MEA)

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



## 9.5 Compute the centroid structure

Collaboration diagram for Compute the centroid structure:



### Functions

- char \* `get_centroid_struct_pl` (int length, double \*dist, plist \*pl)  
Get the centroid structure of the ensemble.
- char \* `get_centroid_struct_pr` (int length, double \*dist, double \*pr)  
Get the centroid structure of the ensemble.

### 9.5.1 Function Documentation

#### 9.5.1.1 char\* get\_centroid\_struct\_pl (int length, double \* dist, plist \* pl)

Get the centroid structure of the ensemble. This function is a threadsafe replacement for `centroid()` with a 'plist' input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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:

- ← *length* The length of the sequence
- *dist* A pointer to the distance variable where the centroid distance will be written to
- ← *pl* A pair list containing base pair probability information about the ensemble

#### Returns:

The centroid structure of the ensemble in dot-bracket notation

#### 9.5.1.2 char\* get\_centroid\_struct\_pr (int length, double \* dist, double \* pr)

Get the centroid structure of the ensemble. This function is a threadsafe replacement for `centroid()` with a probability array input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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 addressed by *dist*.

**Parameters:**

← *length* The length of the sequence

→ *dist* A pointer to the distance variable where the centroid distance will be written to

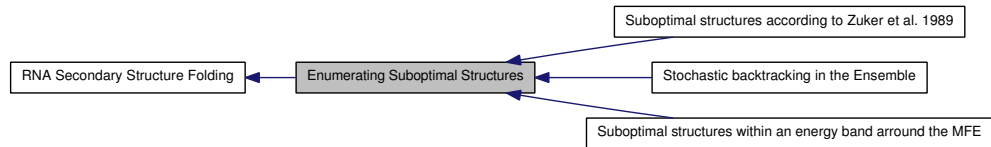
← *pr* A upper triangular matrix containing base pair probabilities (access via `iindx` [get\\_iindx\(\)](#))

**Returns:**

The centroid structure of the ensemble in dot-bracket notation

## 9.6 Enumerating Suboptimal Structures

Collaboration diagram for Enumerating Suboptimal Structures:



### Modules

- [Suboptimal structures according to Zuker et al. 1989](#)
- [Suboptimal structures within an energy band around the MFE](#)
- [Stochastic backtracking in the Ensemble](#)

## 9.7 Suboptimal structures according to Zuker et al. 1989

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



### Functions

- **SOLUTION** \* `zokersubopt` (const char \*string)  
*Compute Zuker type suboptimal structures.*
- **SOLUTION** \* `zokersubopt_par` (const char \*string, paramT \*parameters)  
*Compute Zuker type suboptimal structures.*

### 9.7.1 Function Documentation

#### 9.7.1.1 SOLUTION\* `zokersubopt` (const char \* *string*)

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

#### Parameters:

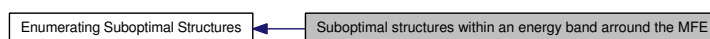
*string* RNA sequence

#### Returns:

List of zuker suboptimal structures

## 9.8 Suboptimal structures within an energy band around the MFE

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



### Functions

- **SOLUTION** \* `subopt` (char \*seq, char \*structure, int delta, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- **SOLUTION** \* `subopt_par` (char \*seq, char \*structure, paramT \*parameters, int delta, int is\_constrained, int is\_circular, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- **SOLUTION** \* `subopt_circ` (char \*seq, char \*sequence, int delta, FILE \*fp)  
*Returns list of circular subopt structures or writes to fp.*

### Variables

- int `subopt_sorted`  
*Sort output by energy.*
- double `print_energy`  
*printing threshold for use with logML*

### 9.8.1 Function Documentation

#### 9.8.1.1 SOLUTION\* subopt (char \* seq, char \* structure, int delta, FILE \* fp)

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

#### Parameters:

*seq*  
*structure*  
*delta*

*fp*

**Returns:**

**9.8.1.2 SOLUTION\* subopt\_circ (char \* *seq*, char \* *sequence*, int *delta*, FILE \* *fp*)**

Returns list of circular subopt structures or writes to *fp*. This function is similar to [subopt\(\)](#) but calculates secondary structures assuming the RNA sequence to be circular instead of linear

**Parameters:**

*seq*

*sequence*

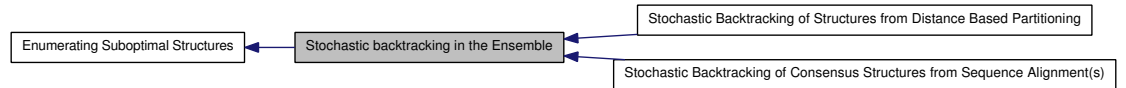
*delta*

*fp*

**Returns:**

## 9.9 Stochastic backtracking in the Ensemble

Collaboration diagram for Stochastic backtracking in the Ensemble:



### Modules

- [Stochastic Backtracking of Consensus Structures from Sequence Alignment\(s\)](#)
- [Stochastic Backtracking of Structures from Distance Based Partitioning](#)

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

### Functions

- `char * pbacktrack (char *sequence)`

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

- `char * pbacktrack\_circ (char *sequence)`

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

### Variables

- `int st\_back`

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

### 9.9.1 Function Documentation

#### 9.9.1.1 `char* pbacktrack (char * sequence)`

Sample a secondary structure from the Boltzmann ensemble according its probability

#### Precondition:

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



**Parameters:**

*sequence* The RNA sequence

**Returns:**

A sampled secondary structure in dot-bracket notation

**9.9.1.2 char\* pbacktrack\_circ (char \* *sequence*)**

Sample a secondary structure of a circular RNA from the Boltzmann ensemble according to its probability. This function does the same as [pbacktrack\(\)](#) but assumes the RNA molecule to be circular

**Precondition:**

[pf\\_fold\\_par\(\)](#) or [pf\\_fold\\_circ\(\)](#) have to be called first to fill the partition function matrices

**Parameters:**

*sequence* The RNA sequence

**Returns:**

A sampled secondary structure in dot-bracket notation

**9.9.2 Variable Documentation****9.9.2.1 int st\_back**

Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking. Set this variable to 1 prior to a call of [pf\\_fold\(\)](#) to ensure that all matrices needed for stochastic backtracking are filled in the forward recursions

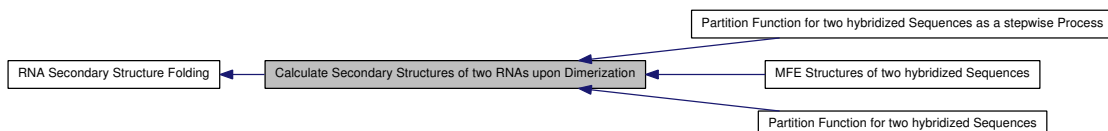
**See also:**

[pbacktrack\(\)](#), [pbacktrack\\_circ](#)

## 9.10 Calculate Secondary Structures of two RNAs upon Dimerization

Predict structures formed by two molecules upon hybridization.

Collaboration diagram for Calculate Secondary Structures of two RNAs upon Dimerization:



### Modules

- [MFE Structures of two hybridized Sequences](#)
- [Partition Function for two hybridized Sequences](#)  
*Partition Function Cofolding.*
- [Partition Function for two hybridized Sequences as a stepwise Process](#)  
*Partition Function Cofolding as a stepwise process.*

### 9.10.1 Detailed Description

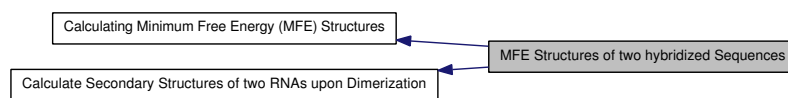
Predict structures formed by two molecules upon hybridization. The function of an RNA molecule often depends on its interaction with other RNAs. The following routines therefore allow to predict structures formed by two RNA molecules upon hybridization.

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

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

## 9.11 MFE Structures of two hybridized Sequences

Collaboration diagram for MFE Structures of two hybridized Sequences:



### Files

- file [cofold.h](#)  
*MFE version of cofolding routines.*

### Functions

- float [cofold](#) (const char \*sequence, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*
- float [cofold\\_par](#) (const char \*string, char \*structure, [paramT](#) \*parameters, int is\_constrained)  
*Compute the minimum free energy of two interacting RNA molecules.*
- void [free\\_co\\_arrays](#) (void)  
*Free memory occupied by [cofold\(\)](#).*
- void [update\\_cofold\\_params](#) (void)  
*Recalculate parameters.*
- void [export\\_cofold\\_arrays\\_gq](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function [cofold](#) (with quadruplex 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](#).*

### 9.11.1 Function Documentation

#### 9.11.1.1 float [cofold](#) (const char \* *sequence*, char \* *structure*)

Compute the minimum free energy of two interacting RNA molecules. The code is analog to the [fold\(\)](#) function. If [cut\\_point](#) == -1 results should be the same as with [fold\(\)](#).

**Parameters:**

*sequence* The two sequences concatenated  
*structure* Will hold the barcket dot structure of the dimer molecule

**Returns:**

minimum free energy of the structure

**9.11.1.2 void export\_cofold\_arrays\_gq (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fMI\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)**

Export the arrays of partition function cofold (with gquadruplex support). Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

**Parameters:**

*f5\_p* A pointer to the 'f5' array, i.e. array conatining best free energy in interval [1,j]  
*c\_p* A pointer to the 'c' array, i.e. array containing best free energy in interval [i,j] given that i pairs with j  
*fML\_p* A pointer to the 'M' array, i.e. array containing best free energy in interval [i,j] for any multiloop segment with at least one stem  
*fMI\_p* A pointer to the 'MI' 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)

**9.11.1.3 void export\_cofold\_arrays (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fMI\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)**

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

**Parameters:**

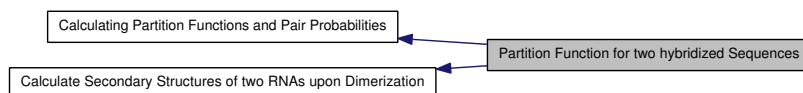
*f5\_p* A pointer to the 'f5' array, i.e. array conatining best free energy in interval [1,j]  
*c\_p* A pointer to the 'c' array, i.e. array containing best free energy in interval [i,j] given that i pairs with j

- fML\_p* A pointer to the 'M' array, i.e. array containing best free energy in interval [i,j] for any multiloop segment with at least one stem
- fM1\_p* A pointer to the 'M1' array, i.e. array containing best free energy in interval [i,j] for multiloop segment with exactly one stem
- fc\_p* A pointer to the 'fc' array, i.e. array ...
- indx\_p* A pointer to the indexing array used for accessing the energy matrices
- pptype\_p* A pointer to the ptype array containing the base pair types for each possibility (i,j)

## 9.12 Partition Function for two hybridized Sequences

Partition Function Cofolding.

Collaboration diagram for Partition Function for two hybridized Sequences:



### Files

- file [part\\_func\\_co.h](#)  
*Partition function for two RNA sequences.*

### Functions

- `cofoldF co_pf_fold` (char \*sequence, char \*structure)  
*Calculate partition function and base pair probabilities.*
- `cofoldF co_pf_fold_par` (char \*sequence, char \*structure, `pf_paramT` \*parameters, int calculate\_bppm, int is\_constrained)  
*Calculate partition function and base pair probabilities.*
- `double * export_co_bppm` (void)  
*Get a pointer to the base pair probability array.*
- `void free_co_pf_arrays` (void)  
*Free the memory occupied by `co_pf_fold()`.*
- `void update_co_pf_params` (int length)  
*Recalculate energy parameters.*
- `void update_co_pf_params_par` (int length, `pf_paramT` \*parameters)  
*Recalculate energy parameters.*
- `void compute_probabilities` (double FAB, double FEA, double FEB, struct `plist` \*prAB, struct `plist` \*prA, struct `plist` \*prB, int Alength)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- `ConcEnt * get_concentrations` (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double \*startconc)  
*Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.*

## Variables

- int [mirnatog](#)  
*Toggles no intrabp in 2nd mol.*
- double [F\\_monomer](#) [2]  
*Free energies of the two monomers.*

### 9.12.1 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 [bernhart:2006](#) for further details.

### 9.12.2 Function Documentation

#### 9.12.2.1 `cofoldF co_pf_fold (char * sequence, char * structure)`

Calculate partition function and base pair probabilities. This is the cofold partition function folding. The second molecule starts at the [cut\\_point](#) nucleotide.

#### Note:

OpenMP: Since this function relies on the global parameters [do\\_backtrack](#), [dangles](#), [temperature](#) and [pf\\_scale](#) it is not threadsafe according to concurrent changes in these variables! Use [co\\_pf\\_fold\\_par\(\)](#) instead to circumvent this issue.

#### See also:

[co\\_pf\\_fold\\_par\(\)](#)

#### Parameters:

*sequence* Concatenated RNA sequences

*structure* Will hold the structure or constraints

#### Returns:

[cofoldF](#) structure containing a set of energies needed for concentration computations.

### 9.12.2.2 `cofoldF co_pf_fold_par (char * sequence, char * structure, pf_paramT * parameters, int calculate_bppm, int is_constrained)`

Calculate partition function and base pair probabilities. This is the cofold partition function folding. The second molecule starts at the [cut\\_point](#) nucleotide.

See also:

[get\\_boltzmann\\_factors\(\)](#), [co\\_pf\\_fold\(\)](#)

Parameters:

*sequence* Concatenated RNA sequences

*structure* Pointer to the structure constraint

*parameters* Data structure containing the precalculated Boltzmann factors

*calculate\_bppm* Switch to turn Base pair probability calculations on/off (0==off)

*is\_constrained* Switch to indicate that a structure constraint is passed via the structure argument (0==off)

Returns:

[cofoldF](#) structure containing a set of energies needed for concentration computations.

### 9.12.2.3 `double* export_co_bppm (void)`

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

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

See also:

[get\\_iindx\(\)](#)

Returns:

A pointer to the base pair probability array

### 9.12.2.4 `void update_co_pf_params (int length)`

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

Note:

This function relies on the global variables [pf\\_scale](#), [dangles](#) and [temperature](#). Thus it might not be threadsafe in certain situations. Use [update\\_co\\_pf\\_params\\_par\(\)](#) instead.



See also:

[get\\_boltzmann\\_factors\(\)](#), [update\\_co\\_pf\\_params\\_par\(\)](#)

Parameters:

*length* Length of the current RNA sequence

#### 9.12.2.5 void update\_co\_pf\_params\_par (int *length*, pf\_paramT \* *parameters*)

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

See also:

[get\\_boltzmann\\_factors\(\)](#), [update\\_co\\_pf\\_params\(\)](#)

Parameters:

*length* Length of the current RNA sequence

*parameters* data structure containing the precomputed Boltzmann factors

#### 9.12.2.6 void compute\_probabilities (double *FAB*, double *FEA*, double *FEB*, struct plist \* *prAB*, struct plist \* *prA*, struct plist \* *prB*, int *Alength*)

Compute Boltzmann probabilities of dimerization without homodimers. Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by [assign\\_plist\\_from\\_pr\(\)](#), the dimer probabilities 'prAB' are modified in place.

Parameters:

*FAB* free energy of dimer AB

*FEA* free energy of monomer A

*FEB* free energy of monomer B

*prAB* pair probabilities for dimer

*prA* pair probabilities monomer

*prB* pair probabilities monomer

*Alength* Length of molecule A

**9.12.2.7 ConcEnt\* get\_concentrations (double *FEAB*, double *FEAA*, double *FEBB*, double *FEA*, double *FEB*, double \* *startconc*)**

Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers. This function takes an array 'startconc' of input concentrations with alternating entries for the initial concentrations of molecules A and B (terminated by two zeroes), then computes the resulting equilibrium concentrations from the free energies for the dimers. Dimer free energies should be the dimer-only free energies, i.e. the FcAB entries from the [cofoldF](#) struct.

**Parameters:**

- FEAB* Free energy of AB dimer (FcAB entry)
- FEAA* Free energy of AA dimer (FcAB entry)
- FEBB* Free energy of BB dimer (FcAB entry)
- FEA* Free energy of monomer A
- FEB* Free energy of monomer B
- startconc* List of start concentrations [a0],[b0],[a1],[b1],..., [an][bn],[0],[0]

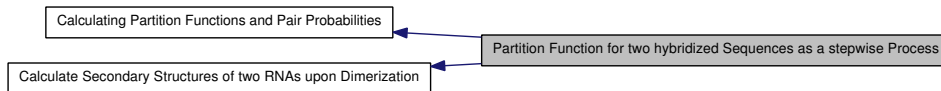
**Returns:**

- [ConcEnt](#) array containing the equilibrium energies and start concentrations

## 9.13 Partition Function for two hybridized Sequences as a stepwise Process

Partition Function Cofolding as a stepwise process.

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



### Files

- file [part\\_func\\_up.h](#)  
*Partition Function Cofolding as stepwise process.*

### Functions

- `pu_contrib * pf_unstru` (`char *sequence`, `int max_w`)  
*Calculate the partition function over all unpaired regions of a maximal length.*
- `interact * pf_interact` (`const char *s1`, `const char *s2`, `pu_contrib *p_c`, `pu_contrib *p_c2`, `int max_w`, `char *cstruc`, `int incr3`, `int incr5`)  
*Calculates the probability of a local interaction between two sequences.*
- `void free_interact` (`interact *pin`)  
*Frees the output of function `pf_interact()`.*
- `void free_pu_contrib_struct` (`pu_contrib *pu`)  
*Frees the output of function `pf_unstru()`.*

#### 9.13.1 Detailed Description

Partition Function Cofolding as a stepwise process.

#### 9.13.2 Function Documentation

##### 9.13.2.1 `pu_contrib* pf_unstru (char * sequence, int max_w)`

Calculate the partition function over all unpaired regions of a maximal length. You have to call function `pf_fold()` providing the same sequence before calling `pf_unstru()`.

If you want to calculate unpaired regions for a constrained structure, set variable 'structure' in function 'pf\_fold()' to the [constrain](#) string. It returns a [pu\\_contrib](#) struct containing four arrays of dimension  $[i = 1 \text{ to } \text{length}(\text{sequence})][j = 0 \text{ to } u-1]$  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:

#### 9.13.2.2 [interact\\*](#) [pf\\_interact](#) ([const char \\*s1](#), [const char \\*s2](#), [pu\\_contrib \\*p\\_c](#), [pu\\_contrib \\*p\\_c2](#), [int max\\_w](#), [char \\*cstruc](#), [int incr3](#), [int incr5](#))

Calculates the probability of a local interaction between two sequences. The function considers the probability that the region of interaction is unpaired within 's1' and 's2'. The longer sequence has to be given as 's1'. The shorter sequence has to be given as 's2'. Function [pf\\_unstru\(\)](#) has to be called for 's1' and 's2', where the probabilities of being unpaired have to be given in 'p\_c' and 'p\_c2', respectively. If you do not want to include the probabilities of being unpaired for 's2' set 'p\_c2' to NULL. If variable 'cstruc' is not NULL, constrained folding is done: The available constrains for intermolecular interaction are: '.' (no [constrain](#)), 'x' (the base has no intermolecular interaction) and '|' (the corresponding base has to be paired intermolecularly).

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  $P_i$  and the minimum free energy in  $G_i$ , where  $i$  is the position in sequence 's1'. The member  $G_{ikjl}$  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'.  $G_{ikjl\_wo}$  is  $G_{ikjl}$  without the probability of being 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*

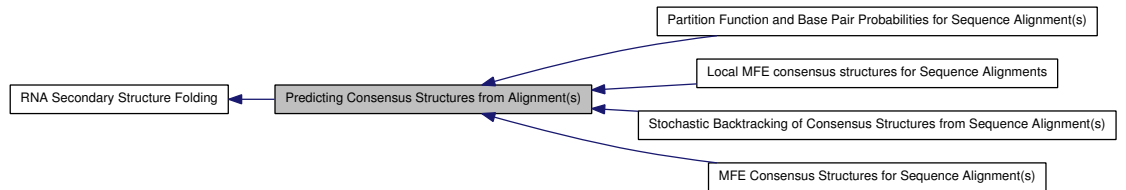
*incr5*

**Returns:**

## 9.14 Predicting Consensus Structures from Alignment(s)

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

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



### Modules

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

### Functions

- `int get\_mpi (char *Aseq[ ], int n_seq, int length, int *mini)`  
*Get the mean pairwise identity in steps from ?to?(ident).*
- `float ** readribosum (char *name)`  
*Read a ribosum or other user-defined scoring matrix.*
- `float energy\_of\_alistruct (const char **sequences, const char *structure, int n_seq, float *energy)`  
*Calculate the free energy of a consensus structure given a set of aligned sequences.*
- `void encode\_ali\_sequence (const char *sequence, short *S, short *s5, short *s3, char *ss, unsigned short *as, int circ)`  
*Get arrays with encoded sequence of the alignment.*
- `void alloc\_sequence\_arrays (const char **sequences, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss, int circ)`  
*Allocate memory for sequence array used to deal with aligned sequences.*
- `void free\_sequence\_arrays (unsigned int n_seq, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss)`  
*Free the memory of the sequence arrays used to deal with aligned sequences.*

- int `get_alipf_arrays` (short `***S_p`, short `***S5_p`, short `***S3_p`, unsigned short `***a2s_p`, char `***Ss_p`, FLT\_OR\_DBL `**qb_p`, FLT\_OR\_DBL `**qm_p`, FLT\_OR\_DBL `**q1k_p`, FLT\_OR\_DBL `**q1n_p`, short `**pscore`)

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

## Variables

- double `cv_fact`

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

- double `nc_fact`

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

### 9.14.1 Detailed Description

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

### 9.14.2 Function Documentation

#### 9.14.2.1 `int get_mpi(char * Aseq[], int n_seq, int length, int * mini)`

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

#### Parameters:

*Aseq*

*n\_seq* The number of sequences in the alignment

*length* The length of the alignment

*mini*

#### Returns:

The mean pairwise identity

### 9.14.2.2 `float energy_of_alistruct (const char ** sequences, const char * structure, int n_seq, float * energy)`

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

#### Parameters:

*sequences* The NULL terminated array of sequences

*structure* The consensus structure

*n\_seq* The number of sequences in the alignment

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

#### Returns:

free energy in kcal/mol

### 9.14.2.3 `void encode_ali_sequence (const char * sequence, short * S, short * s5, short * s3, char * ss, unsigned short * as, int circ)`

Get arrays with encoded sequence of the alignment. this function assumes that in S, S5, s3, ss and as enough space is already allocated (size must be at least sequence length+2)

#### Parameters:

*sequence* The gapped sequence from the alignment

*S* pointer to an array that holds encoded sequence

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

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

*ss*

*as*

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

### 9.14.2.4 `void alloc_sequence_arrays (const char ** sequences, short *** S, short *** S5, short *** S3, unsigned short *** a2s, char *** Ss, int circ)`

Allocate memory for sequence array used to deal with aligned sequences. Note that these arrays will also be initialized according to the sequence alignment given

#### See also:

[free\\_sequence\\_arrays\(\)](#)



**Parameters:**

- sequences* The aligned sequences
- S* A pointer to the array of encoded sequences
- S5* A pointer to the array that contains the next 5' nucleotide of a sequence position
- S3* A pointer to the array that contains the next 3' nucleotide of a sequence position
- a2s* A pointer to the array that contains the alignment to sequence position mapping
- Ss* A pointer to the array that contains the ungapped sequence
- circ* assume the molecules to be circular instead of linear (circ=0)

**9.14.2.5 void free\_sequence\_arrays (unsigned int *n\_seq*, short \*\*\* *S*, short \*\*\* *S5*, short \*\*\* *S3*, unsigned short \*\*\* *a2s*, char \*\*\* *Ss*)**

Free the memory of the sequence arrays used to deal with aligned sequences. This function frees the memory previously allocated with [alloc\\_sequence\\_arrays\(\)](#)

**See also:**

[alloc\\_sequence\\_arrays\(\)](#)

**Parameters:**

- n\_seq* The number of aligned sequences
- S* A pointer to the array of encoded sequences
- S5* A pointer to the array that contains the next 5' nucleotide of a sequence position
- S3* A pointer to the array that contains the next 3' nucleotide of a sequence position
- a2s* A pointer to the array that contains the alignment to sequence position mapping
- Ss* A pointer to the array that contains the ungapped sequence

**9.14.2.6 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.

**Note:**

To obtain meaningful pointers, call `alipf_fold` first!

**See also:**

`pf_alifold()`, [alipf\\_circ\\_fold\(\)](#)

**Parameters:**

*S\_p* A pointer to the 'S' array (integer representation of nucleotides)

*S5\_p* A pointer to the 'S5' array

*S3\_p* A pointer to the 'S3' array

*a2s\_p* A pointer to the pair type matrix

*Ss\_p* A pointer to the 'Ss' array

*qb\_p* A pointer to the  $Q^B$  matrix

*qm\_p* A pointer to the  $Q^M$  matrix

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

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

**Returns:**

Non Zero if everything went fine, 0 otherwise

### 9.14.3 Variable Documentation

#### 9.14.3.1 double cv\_fact

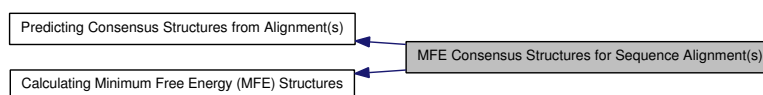
This variable controls the weight of the covariance term in the energy function of alignment folding algorithms. Default is 1.

#### 9.14.3.2 double nc\_fact

This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms. Default is 1.

## 9.15 MFE Consensus Structures for Sequence Alignment(s)

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



### Functions

- float [alifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according consensus structure of an alignment of sequences.*
- float [circularifold](#) (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.*

### 9.15.1 Function Documentation

#### 9.15.1.1 float alifold (const char \*\* strings, char \* structure)

Compute MFE and according consensus structure of an alignment of sequences. This function predicts the consensus structure for the aligned 'sequences' and returns the minimum free energy; the mfe structure in bracket notation is returned in 'structure'.

Sufficient space must be allocated for 'structure' before calling [alifold\(\)](#).

#### Parameters:

*strings* A pointer to a NULL terminated array of character arrays

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

#### Returns:

The free energy score in kcal/mol

#### 9.15.1.2 float circularifold (const char \*\* strings, char \* structure)

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

**Parameters:**

*strings* A pointer to a NULL terminated array of character arrays

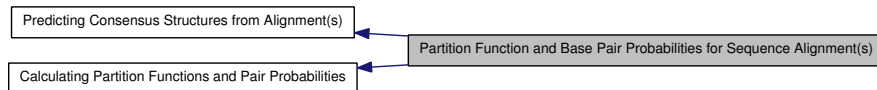
*structure* A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

**Returns:**

The free energy score in kcal/mol

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

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



### Functions

- float `alipf_fold_par` (const char \*\*sequences, char \*structure, `plist` \*\*pl, `pf_paramT` \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)
- float `alipf_fold` (const char \*\*sequences, char \*structure, `plist` \*\*pl)

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

- float `alipf_circ_fold` (const char \*\*sequences, char \*structure, `plist` \*\*pl)
- FLT\_OR\_DBL \* `export_ali_bppm` (void)

*Get a pointer to the base pair probability array.*

### 9.16.1 Function Documentation

**9.16.1.1** float `alipf_fold_par` (const char \*\* sequences, char \* structure, `plist` \*\* pl, `pf_paramT` \* parameters, int calculate\_bppm, int is\_constrained, int is\_circular)

#### Parameters:

*sequences*

*structure*

*pl*

*parameters*

*calculate\_bppm*

*is\_constrained*

*is\_circular*

#### Returns:

### 9.16.1.2 float `alipf_fold` (`const char ** sequences`, `char * structure`, `plist ** pl`)

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

**Parameters:**

*sequences*

*structure*

*pl*

**Returns:**

### 9.16.1.3 float `alipf_circ_fold` (`const char ** sequences`, `char * structure`, `plist ** pl`)

**Parameters:**

*sequences*

*structure*

*pl*

**Returns:**

### 9.16.1.4 `FLT_OR_DBL* export_ali_bppm` (`void`)

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

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

**See also:**

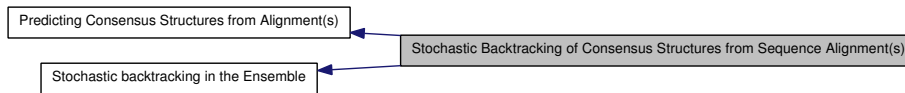
[get\\_iindx\(\)](#)

**Returns:**

A pointer to the base pair probability array

## 9.17 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)

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



### Functions

- `char * alipbacktrack (double *prob)`

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

.

### 9.17.1 Function Documentation

#### 9.17.1.1 `char* alipbacktrack (double * prob)`

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

.

#### Parameters:

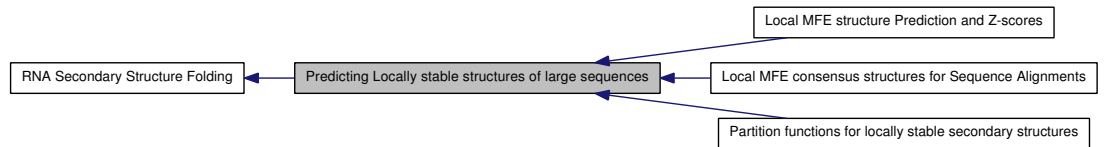
*prob* to be described (berni)

#### Returns:

A sampled consensus secondary structure in dot-bracket notation

## 9.18 Predicting Locally stable structures of large sequences

Collaboration diagram for Predicting Locally stable structures of large sequences:



### Modules

- [Local MFE structure Prediction and Z-scores](#)
- [Partition functions for locally stable secondary structures](#)
- [Local MFE consensus structures for Sequence Alignments](#)

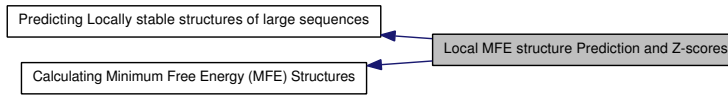
#### 9.18.1 Detailed Description

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



## 9.19 Local MFE structure Prediction and Z-scores

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



### Functions

- float **Lfold** (const char \*string, char \*structure, int maxdist)  
*The local analog to [fold\(\)](#).*
- float **Lfoldz** (const char \*string, char \*structure, int maxdist, int zsc, double min\_z)

#### 9.19.1 Function Documentation

##### 9.19.1.1 float Lfold (const char \* string, char \* structure, int maxdist)

The local analog to [fold\(\)](#). Computes the minimum free energy structure including only base pairs with a span smaller than 'maxdist'

##### Parameters:

*string*  
*structure*  
*maxdist*

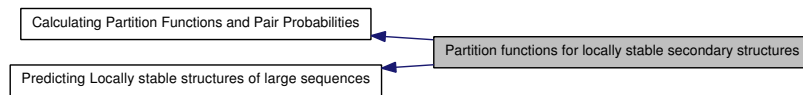
##### 9.19.1.2 float Lfoldz (const char \* string, char \* structure, int maxdist, int zsc, double min\_z)

##### Parameters:

*string*  
*structure*  
*maxdist*  
*zsc*  
*min\_z*

## 9.20 Partition functions for locally stable secondary structures

Collaboration diagram for Partition functions for locally stable secondary structures:



### Functions

- void `update_pf_paramsLP` (int length)
- `plist * pfl_fold` (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, struct `plist` \*\*dpp2, FILE \*pUfp, FILE \*spup)  
*Compute partition functions for locally stable secondary structures.*
- `plist * pfl_fold_par` (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, struct `plist` \*\*dpp2, FILE \*pUfp, FILE \*spup, `pf_paramT` \*parameters)  
*Compute partition functions for locally stable secondary structures.*
- void `putoutpU_prob` (double \*\*pU, int length, int ulength, FILE \*fp, int energies)  
*Writes the unpaired probabilities (pU) or opening energies into a file.*
- void `putoutpU_prob_bin` (double \*\*pU, int length, int ulength, FILE \*fp, int energies)  
*Writes the unpaired probabilities (pU) or opening energies into a binary file.*

### 9.20.1 Function Documentation

#### 9.20.1.1 void update\_pf\_paramsLP (int length)

Parameters:

*length*

#### 9.20.1.2 `plist* pfl_fold` (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, struct `plist` \*\*dpp2, FILE \*pUfp, FILE \*spup)

Compute partition functions for locally stable secondary structures. `pfl_fold` computes partition functions for every window of size 'winSize' possible in a RNA molecule, allowing only pairs with a span smaller than 'pairSize'. It returns the mean pair probabilities averaged over all windows containing the pair in 'pl'. 'winSize' should always

be  $\geq$  '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 immediately. If spup is given (i.e. is not NULL), the pair probabilities in pl are not saved but put out immediately.

#### 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

#### 9.20.1.3 void putoutU\_prob (double \*\* pU, int length, int ulength, FILE \* fp, int energies)

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

#### Parameters:

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

**9.20.1.4 void putoutpU\_prob\_bin (double \*\* *pU*, int *length*, int *ulength*, FILE \**fp*, int *energies*)**

Writes the unpaired probabilities (pU) or opening energies into a binary file. Can write either the unpaired probabilities (accessibilities) pU or the opening energies  $-\log(pU)kT$  into a file

**Parameters:**

*pU* pair probabilities

*length* length of RNA sequence

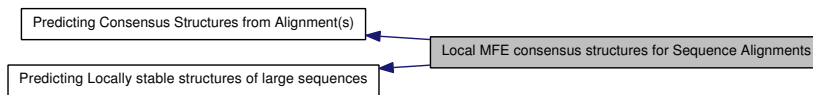
*ulength* maximum length of unpaired stretch

*fp* file pointer of destination file

*energies* switch to put out as opening energies

## 9.21 Local MFE consensus structures for Sequence Alignments

Collaboration diagram for Local MFE consensus structures for Sequence Alignments:



### Functions

- float [aliLfold](#) (const char \*\*strings, char \*structure, int maxdist)

#### 9.21.1 Function Documentation

##### 9.21.1.1 float aliLfold (const char \*\* strings, char \* structure, int maxdist)

###### Parameters:

*strings*

*structure*

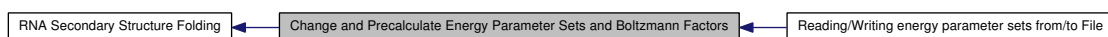
*maxdist*

###### Returns:

## 9.22 Change and Precalculate Energy Parameter Sets and Boltzmann Factors

All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

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



### Modules

- [Reading/Writing energy parameter sets from/to File](#)

*Read and Write energy parameter sets from and to text files.*

### Files

- file [params.h](#)

### Functions

- `paramT * scale_parameters` (void)  
*Get precomputed energy contributions for all the known loop types.*
- `paramT * get_scaled_parameters` (double `temperature`, `model_detailsT` md)  
*Get precomputed energy contributions for all the known loop types.*
- `pf_paramT * get_scaled_pf_parameters` (void)
- `pf_paramT * get_boltzmann_factors` (double `temperature`, double `betaScale`, `model_detailsT` md, double `pf_scale`)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.*
- `pf_paramT * get_boltzmann_factor_copy` (`pf_paramT *parameters`)  
*Get a copy of already precomputed Boltzmann factors.*
- `pf_paramT * get_scaled_alipf_parameters` (unsigned int `n_seq`)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant).*
- `pf_paramT * get_boltzmann_factors_ali` (unsigned int `n_seq`, double `temperature`, double `betaScale`, `model_detailsT` md, double `pf_scale`)

## **9.22 Change and Precalculate Energy Parameter Sets and Boltzmann Factors 83**

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

### **9.22.1 Detailed Description**

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

### **9.22.2 Function Documentation**

#### **9.22.2.1 paramT\* scale\_parameters (void)**

Get precomputed energy contributions for all the known loop types.

**Note:**

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [get\\_scaled\\_parameters\(\)](#) for a completely threadsafe implementation.

**Returns:**

A set of precomputed energy contributions

#### **9.22.2.2 paramT\* get\_scaled\_parameters (double *temperature*, model\_detailsT *md*)**

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

**See also:**

[model\\_detailsT](#), [set\\_model\\_details\(\)](#)

**Parameters:**

*temperature* The temperature in degrees Celcius

*md* The model details

**Returns:**

precomputed energy contributions and model settings

### 9.22.2.3 `pf_paramT*` `get_scaled_pf_parameters` (void)

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

#### Returns:

The datastructure containing Boltzmann weights for use in partition function calculations

### 9.22.2.4 `pf_paramT*` `get_boltzmann_factors` (double *temperature*, double *betaScale*, `model_detailsT` *md*, double *pf\_scale*)

Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature. This function returns a data structure that contains all necessary precalculated Boltzmann factors for each loop type contribution.

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

#### See also:

[get\\_scaled\\_pf\\_parameters\(\)](#), [get\\_boltzmann\\_factor\\_copy\(\)](#)

#### Parameters:

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

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

*md* The model details to be used

*pf\_scale* The scaling factor for the Boltzmann factors

#### Returns:

A set of precomputed Boltzmann factors

### 9.22.2.5 `pf_paramT*` `get_boltzmann_factor_copy` (`pf_paramT` \* *parameters*)

Get a copy of already precomputed Boltzmann factors.

#### See also:

[get\\_boltzmann\\_factors\(\)](#), [get\\_scaled\\_pf\\_parameters\(\)](#)

#### Parameters:

*parameters* The input data structure that shall be copied



## **9.22 Change and Precalculate Energy Parameter Sets and Boltzmann Factors 85**

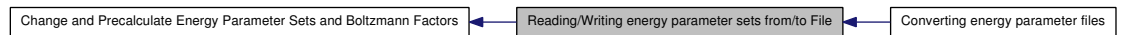
### **Returns:**

A copy of the provided Boltzmann factor dataset

## 9.23 Reading/Writing energy parameter sets from/to File

Read and Write energy parameter sets from and to text files.

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



### Modules

- [Converting energy parameter files](#)  
*Convert energy parameter files into the latest format.*

### Files

- file [read\\_epars.h](#)

### Functions

- void [read\\_parameter\\_file](#) (const char fname[ ])  
*Read energy parameters from a file.*
- void [write\\_parameter\\_file](#) (const char fname[ ])  
*Write energy parameters to a file.*

#### 9.23.1 Detailed Description

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

#### 9.23.2 Function Documentation

##### 9.23.2.1 void read\_parameter\_file (const char fname[ ])

Read energy parameters from a file.

##### Parameters:

*fname* The path to the file containing the energy parameters

9.23.2.2 void write\_parameter\_file (const char *fname* [ ])

Write energy parameters to a file.

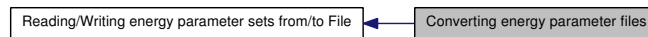
**Parameters:**

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

## 9.24 Converting energy parameter files

Convert energy parameter files into the latest format.

Collaboration diagram for Converting energy parameter files:



### Files

- file [convert\\_epars.h](#)

*Functions and definitions for energy parameter file format conversion.*

### Defines

- #define [VRNA\\_CONVERT\\_OUTPUT\\_ALL](#) 1U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_HP](#) 2U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_STACK](#) 4U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_HP](#) 8U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT](#) 16U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT\\_1N](#) 32U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT\\_23](#) 64U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_MULTI](#) 128U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_EXT](#) 256U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_DANGLE5](#) 512U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_DANGLE3](#) 1024U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_INT\\_11](#) 2048U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_INT\\_21](#) 4096U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_INT\\_22](#) 8192U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_BULGE](#) 16384U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_INT](#) 32768U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_ML](#) 65536U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MISC](#) 131072U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_SPECIAL\\_HP](#) 262144U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_VANILLA](#) 524288U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_NINIO](#) 1048576U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_DUMP](#) 2097152U

### Functions

- void [convert\\_parameter\\_file](#) (const char \*iname, const char \*oname, unsigned int options)

### 9.24.1 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

### 9.24.2 Define Documentation

#### 9.24.2.1 `#define VRNA_CONVERT_OUTPUT_ALL 1U`

Flag to indicate printing of a complete parameter set

#### 9.24.2.2 `#define VRNA_CONVERT_OUTPUT_HP 2U`

Flag to indicate printing of hairpin contributions

#### 9.24.2.3 `#define VRNA_CONVERT_OUTPUT_STACK 4U`

Flag to indicate printing of base pair stack contributions

#### 9.24.2.4 `#define VRNA_CONVERT_OUTPUT_MM_HP 8U`

Flag to indicate printing of hairpin mismatch contribution

#### 9.24.2.5 `#define VRNA_CONVERT_OUTPUT_MM_INT 16U`

Flag to indicate printing of interior loop mismatch contribution

#### 9.24.2.6 `#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U`

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

#### 9.24.2.7 `#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U`

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

#### 9.24.2.8 `#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U`

Flag to indicate printing of multi loop mismatch contribution

#### 9.24.2.9 `#define VRNA_CONVERT_OUTPUT_MM_EXT 256U`

Flag to indicate printing of exterior loop mismatch contribution

**9.24.2.10 #define VRNA\_CONVERT\_OUTPUT\_DANGLE5 512U**

Flag to indicate printing of 5' dangle contribution

**9.24.2.11 #define VRNA\_CONVERT\_OUTPUT\_DANGLE3 1024U**

Flag to indicate printing of 3' dangle contribution

**9.24.2.12 #define VRNA\_CONVERT\_OUTPUT\_INT\_11 2048U**

Flag to indicate printing of 1:1 interior loop contribution

**9.24.2.13 #define VRNA\_CONVERT\_OUTPUT\_INT\_21 4096U**

Flag to indicate printing of 2:1 interior loop contribution

**9.24.2.14 #define VRNA\_CONVERT\_OUTPUT\_INT\_22 8192U**

Flag to indicate printing of 2:2 interior loop contribution

**9.24.2.15 #define VRNA\_CONVERT\_OUTPUT\_BULGE 16384U**

Flag to indicate printing of bulge loop contribution

**9.24.2.16 #define VRNA\_CONVERT\_OUTPUT\_INT 32768U**

Flag to indicate printing of interior loop contribution

**9.24.2.17 #define VRNA\_CONVERT\_OUTPUT\_ML 65536U**

Flag to indicate printing of multi loop contribution

**9.24.2.18 #define VRNA\_CONVERT\_OUTPUT\_MISC 131072U**

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

**9.24.2.19 #define VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP 262144U**

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

**9.24.2.20 #define VRNA\_CONVERT\_OUTPUT\_VANILLA 524288U**

Flag to indicate printing of given parameters only

**Note:**

This option overrides all other output options, except `VRNA_CONVERT_OUTPUT_DUMP` !

**9.24.2.21 #define VRNA\_CONVERT\_OUTPUT\_NINIO 1048576U**

Flag to indicate printing of interior loop asymmetry contribution

**9.24.2.22 #define VRNA\_CONVERT\_OUTPUT\_DUMP 2097152U**

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

**9.24.3 Function Documentation****9.24.3.1 void convert\_parameter\_file (const char \* *iname*, const char \* *oname*, unsigned int *options*)**

Convert/dump a Vienna 1.8.4 formatted energy parameter file

The options argument allows to control the different output modes.

Currently available options are:

`VRNA_CONVERT_OUTPUT_ALL`, `VRNA_CONVERT_OUTPUT_HP`, `VRNA_CONVERT_OUTPUT_STACK`

`VRNA_CONVERT_OUTPUT_MM_HP`, `VRNA_CONVERT_OUTPUT_MM_INT`, `VRNA_CONVERT_OUTPUT_MM_INT_1N`

`VRNA_CONVERT_OUTPUT_MM_INT_23`, `VRNA_CONVERT_OUTPUT_MM_MULTI`, `VRNA_CONVERT_OUTPUT_MM_EXT`

`VRNA_CONVERT_OUTPUT_DANGLE5`, `VRNA_CONVERT_OUTPUT_DANGLE3`, `VRNA_CONVERT_OUTPUT_INT_11`

`VRNA_CONVERT_OUTPUT_INT_21`, `VRNA_CONVERT_OUTPUT_INT_22`, `VRNA_CONVERT_OUTPUT_BULGE`

`VRNA_CONVERT_OUTPUT_INT`, `VRNA_CONVERT_OUTPUT_ML`, `VRNA_CONVERT_OUTPUT_MISC`

`VRNA_CONVERT_OUTPUT_SPECIAL_HP`, `VRNA_CONVERT_OUTPUT_VANILLA`, `VRNA_CONVERT_OUTPUT_NINIO`

`VRNA_CONVERT_OUTPUT_DUMP`

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

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

**Parameters:**

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

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

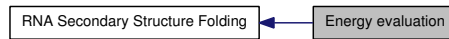
*options* The options (as described above)



## 9.25 Energy evaluation

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

Collaboration diagram for Energy evaluation:



### Functions

- float `energy_of_structure` (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded RNA using global model detail settings.*
- float `energy_of_struct_par` (const char \*string, const char \*structure, paramT \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float `energy_of_circ_structure` (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- float `energy_of_circ_struct_par` (const char \*string, const char \*structure, paramT \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- int `energy_of_structure_pt` (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- int `energy_of_struct_pt_par` (const char \*string, short \*ptable, short \*s, short \*s1, paramT \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*

### Variables

- int `eos_debug`  
*verbose info from energy\_of\_struct*

#### 9.25.1 Detailed Description

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

## 9.25.2 Function Documentation

### 9.25.2.1 float `energy_of_structure` (`const char * string`, `const char * structure`, `int verbosity_level`)

Calculate the free energy of an already folded RNA using global model detail settings. If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**Note:**

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [energy\\_of\\_struct\\_par\(\)](#) for a completely threadsafe implementation.

**See also:**

[energy\\_of\\_struct\\_par\(\)](#), [energy\\_of\\_circ\\_structure\(\)](#)

**Parameters:**

*string* RNA sequence  
*structure* secondary structure in dot-bracket notation  
*verbosity\_level* a flag to turn verbose output on/off

**Returns:**

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

### 9.25.2.2 float `energy_of_struct_par` (`const char * string`, `const char * structure`, `paramT * parameters`, `int verbosity_level`)

Calculate the free energy of an already folded RNA. If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**See also:**

[energy\\_of\\_circ\\_structure\(\)](#), [energy\\_of\\_structure\\_pt\(\)](#), [get\\_scaled\\_parameters\(\)](#)

**Parameters:**

*string* RNA sequence in uppercase letters  
*structure* Secondary structure in dot-bracket notation  
*parameters* A data structure containing the prescaled energy contributions and the model details.  
*verbosity\_level* A flag to turn verbose output on/off

**Returns:**

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

**9.25.2.3 float energy\_of\_circ\_structure (const char \* *string*, const char \* *structure*, int *verbosity\_level*)**

Calculate the free energy of an already folded circular RNA.

**Note:**

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [energy\\_of\\_circ\\_struct\\_par\(\)](#) for a completely threadsafe implementation.

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

**See also:**

[energy\\_of\\_circ\\_struct\\_par\(\)](#), [energy\\_of\\_struct\\_par\(\)](#)

**Parameters:**

*string* RNA sequence

*structure* Secondary structure in dot-bracket notation

*verbosity\_level* A flag to turn verbose output on/off

**Returns:**

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

**9.25.2.4 float energy\_of\_circ\_struct\_par (const char \* *string*, const char \* *structure*, paramT \* *parameters*, int *verbosity\_level*)**

Calculate the free energy of an already folded circular RNA. If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**See also:**

[energy\\_of\\_struct\\_par\(\)](#), [get\\_scaled\\_parameters\(\)](#)

**Parameters:**

*string* RNA sequence

*structure* Secondary structure in dot-bracket notation

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

*verbosity\_level* A flag to turn verbose output on/off

**Returns:**

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

### 9.25.2.5 `int energy_of_structure_pt (const char * string, short * ptable, short * s, short * sI, int verbosity_level)`

Calculate the free energy of an already folded RNA. If verbosity level is set to a value >0, energies of structure elements are printed to stdout

#### Note:

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [energy\\_of\\_struct\\_pt\\_par\(\)](#) for a completely threadsafe implementation.

#### See also:

[make\\_pair\\_table\(\)](#), [energy\\_of\\_struct\\_pt\\_par\(\)](#)

#### Parameters:

*string* RNA sequence  
*ptable* the pair table of the secondary structure  
*s* encoded RNA sequence  
*sI* 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

### 9.25.2.6 `int energy_of_struct_pt_par (const char * string, short * ptable, short * s, short * sI, paramT * parameters, int verbosity_level)`

Calculate the free energy of an already folded RNA. If verbosity level is set to a value >0, energies of structure elements are printed to stdout

#### See also:

[make\\_pair\\_table\(\)](#), [energy\\_of\\_struct\\_par\(\)](#), [get\\_scaled\\_parameters\(\)](#)

#### Parameters:

*string* RNA sequence in uppercase letters  
*ptable* The pair table of the secondary structure  
*s* Encoded RNA sequence  
*sI* 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

## 9.26 Searching Sequences for Predefined Structures

Collaboration diagram for Searching Sequences for Predefined Structures:



### Files

- file [inverse.h](#)  
*Inverse folding routines.*

### Functions

- float [inverse\\_fold](#) (char \*start, const char \*target)  
*Find sequences with predefined structure.*
- float [inverse\\_pf\\_fold](#) (char \*start, const char \*target)  
*Find sequence that maximizes probability of a predefined structure.*

### Variables

- char \* [symbolset](#)  
*This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.*
- float [final\\_cost](#)
- int [give\\_up](#)
- int [inv\\_verbose](#)

#### 9.26.1 Detailed Description

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

#### 9.26.2 Function Documentation

##### 9.26.2.1 float [inverse\\_fold](#) (char \* *start*, const char \* *target*)

Find sequences with predefined structure. This function searches for a sequence with minimum free energy structure provided in the parameter 'target', starting with sequence 'start'. It returns 0 if the search was successful, otherwise a structure distance

in terms of the energy difference between the search result and the actual target 'target' is returned. The found sequence is returned in 'start'. If [give\\_up](#) is set to 1, the function will return as soon as it is clear that the search will be unsuccessful, this speeds up the algorithm if you are only interested in exact solutions.

**Parameters:**

*start* The start sequence

*target* The target secondary structure in dot-bracket notation

**Returns:**

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

**9.26.2.2 float inverse\_pf\_fold (char \* start, const char \* target)**

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

**Parameters:**

*start* The start sequence

*target* The target secondary structure in dot-bracket notation

**Returns:**

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

**9.26.3 Variable Documentation****9.26.3.1 float final\_cost**

when to stop [inverse\\_pf\\_fold\(\)](#)

**9.26.3.2 int give\_up**

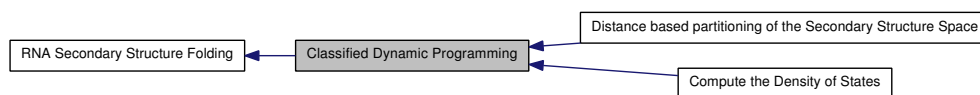
default 0: try to minimize structure distance even if no exact solution can be found

**9.26.3.3 int inv\_verbose**

print out substructure on which [inverse\\_fold\(\)](#) fails

## 9.27 Classified Dynamic Programming

Collaboration diagram for Classified Dynamic Programming:



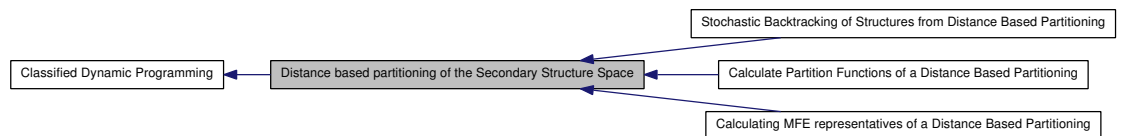
### Modules

- [Distance based partitioning of the Secondary Structure Space](#)  
*Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.*
- [Compute the Density of States](#)

## 9.28 Distance based partitioning of the Secondary Structure Space

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

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.*

### 9.28.1 Detailed Description

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

#### See also:

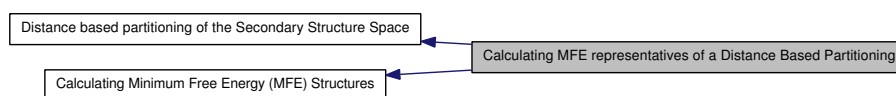
For further details have a look into [lorenz:2009](#)



## 9.29 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.

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



### Files

- file [2Dfold.h](#)

### Functions

- `TwoDfold_vars` \* [get\\_TwoDfold\\_variables](#) (const char \*seq, const char \*structure1, const char \*structure2, int circ)

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

- void [destroy\\_TwoDfold\\_variables](#) (`TwoDfold_vars` \*our\_variables)

*Destroy a `TwoDfold_vars` datastructure without memory loss.*

- `TwoDfold_solution` \* [TwoDfoldList](#) (`TwoDfold_vars` \*vars, int distance1, int distance2)

*Compute MFE's and representative for distance partitioning.*

- char \* [TwoDfold\\_backtrack\\_f5](#) (unsigned int j, int k, int l, `TwoDfold_vars` \*vars)

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

### 9.29.1 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.

## 9.29.2 Function Documentation

### 9.29.2.1 `TwoDfold_vars*` `get_TwoDfold_variables` (`const char * seq`, `const char * structure1`, `const char * structure2`, `int circ`)

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

#### Note:

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

#### See also:

[destroy\\_TwoDfold\\_variables\(\)](#), [TwoDfold\(\)](#), [TwoDfold\\_circ](#)

#### Parameters:

*seq* The RNA sequence  
*structure1* The first reference structure in dot-bracket notation  
*structure2* The second reference structure in dot-bracket notation  
*circ* A switch to indicate the assumption to fold a circular instead of linear RNA (0=OFF, 1=ON)

#### Returns:

A datastructure prefilled with folding options and allocated memory

### 9.29.2.2 `void` `destroy_TwoDfold_variables` (`TwoDfold_vars * our_variables`)

Destroy a `TwoDfold_vars` datastructure without memory loss. This function free's all allocated memory that depends on the datastructure given.

#### See also:

[get\\_TwoDfold\\_variables\(\)](#)

#### Parameters:

*our\_variables* A pointer to the datastructure to be destroyed

### 9.29.2.3 `TwoDfold_solution*` `TwoDfoldList` (`TwoDfold_vars * vars`, `int distance1`, `int distance2`)

Compute MFE's and representative for distance partitioning. This function computes the minimum free energies and a representative secondary structure for each distance

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

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

**See also:**

[get\\_TwoDfold\\_variables\(\)](#), [destroy\\_TwoDfold\\_variables\(\)](#), [TwoDfold\\_solution](#)

**Parameters:**

*vars* the datastructure containing all predefined folding attributes  
*distance1* maximum distance to reference1 (-1 means no restriction)  
*distance2* maximum distance to reference2 (-1 means no restriction)

**9.29.2.4 char\* TwoDfold\_backtrack\_f5 (unsigned int j, int k, int l, TwoDfold\_vars \* vars)**

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

**Note:**

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

**See also:**

[TwoDfoldList\(\)](#), [get\\_TwoDfold\\_variables\(\)](#), [destroy\\_TwoDfold\\_variables\(\)](#)

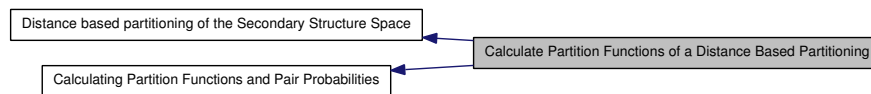
**Parameters:**

*j* The length in nucleotides beginning from the 5' end  
*k* distance to reference1 (may be -1)  
*l* distance to reference2  
*vars* the datastructure containing all predefined folding attributes

## 9.30 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.

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



### Files

- file [2Dpfold.h](#)

### Functions

- `TwoDpfold_vars * get_TwoDpfold_variables` (const char \*seq, const char \*structure1, char \*structure2, int circ)  
*Get a datastructure containing all necessary attributes and global folding switches.*
- `TwoDpfold_vars * get_TwoDpfold_variables_from_MFE` (TwoDfold\_vars \*mfe\_vars)  
*Get the datastructure containing all necessary attributes and global folding switches from a pre-filled mfe-datastructure.*
- void `destroy_TwoDpfold_variables` (TwoDpfold\_vars \*vars)  
*Free all memory occupied by a TwoDpfold\_vars datastructure.*
- `TwoDpfold_solution * TwoDpfoldList` (TwoDpfold\_vars \*vars, int maxDistance1, int maxDistance2)  
*Compute the partition function for all distance classes.*

#### 9.30.1 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.

## 9.30.2 Function Documentation

### 9.30.2.1 `TwoDpfold_vars* get_TwoDpfold_variables (const char * seq, const char * structure1, char * structure2, int circ)`

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

#### Parameters:

- seq* the RNA sequence in uppercase format with letters from the alphabet {AUCG}
- structure1* the first reference structure in dot-bracket notation
- structure2* the second reference structure in dot-bracket notation
- circ* a switch indicating if the sequence is linear (0) or circular (1)

#### Returns:

the datastructure containing all necessary partition function attributes

### 9.30.2.2 `TwoDpfold_vars* get_TwoDpfold_variables_from_MFE (TwoDfold_vars * mfe_vars)`

Get the datastructure containing all necessary attributes and global folding switches from a pre-filled `mfe`-datastructure. This function actually does the same as `get_TwoDpfold_variables` but takes its switches and settings from a pre-filled MFE equivalent datastructure

#### See also:

[get\\_TwoDfold\\_variables\(\)](#), [get\\_TwoDpfold\\_variables\(\)](#)

#### Parameters:

- mfe\_vars* the pre-filled `mfe` datastructure

#### Returns:

the datastructure containing all necessary partition function attributes

### 9.30.2.3 `void destroy_TwoDpfold_variables (TwoDpfold_vars * vars)`

Free all memory occupied by a `TwoDpfold_vars` datastructure. This function free's all memory occupied by a datastructure obtained from from [get\\_TwoDpfold\\_variables\(\)](#) or [get\\_TwoDpfold\\_variables\\_from\\_MFE\(\)](#)

**See also:**

[get\\_TwoDpfold\\_variables\(\)](#), [get\\_TwoDpfold\\_variables\\_from\\_MFE\(\)](#)

**Parameters:**

*vars* the datastructure to be free'd

**9.30.2.4 TwoDpfold\_solution\* TwoDpfoldList (TwoDpfold\_vars \* vars, int maxDistance1, int maxDistance2)**

Compute the partition function for all distance classes. This function computes the partition functions for all distance classes according the two reference structures specified in the datastructure 'vars'. Similar to [TwoDfoldList\(\)](#) the arguments maxDistance1 and maxDistance2 specify the maximum distance to both reference structures. A value of '-1' in either of them makes the appropriate distance restrictionless, i.e. all basepair distances 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=-1 contains the partition function for all structures exceeding the restriction. A values of [INF](#) in the attribute 'k' of the returned list denotes the end of the list

**See also:**

[get\\_TwoDpfold\\_variables\(\)](#), [destroy\\_TwoDpfold\\_variables\(\)](#), [TwoDpfold\\_solution](#)

**Parameters:**

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

*maxDistance1* the maximum basepair distance to reference1 (may be -1)

*maxDistance2* the maximum basepair distance to reference2 (may be -1)

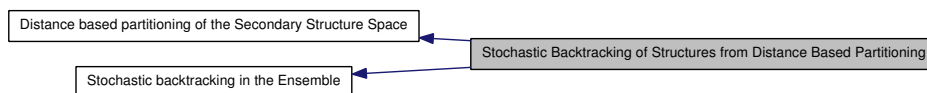
**Returns:**

a list of partition funtions for the appropriate distance classes

### 9.31 Stochastic Backtracking of Structures from Distance Based Partitioning

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

Collaboration diagram for Stochastic Backtracking of Structures from Distance Based Partitioning:



#### Functions

- `char * TwoDpfold_pbacktrack (TwoDpfold_vars *vars, int d1, int d2)`  
*Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- `char * TwoDpfold_pbacktrack5 (TwoDpfold_vars *vars, int d1, int d2, unsigned int length)`  
*Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.*

#### 9.31.1 Detailed Description

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

#### 9.31.2 Function Documentation

##### 9.31.2.1 `char* TwoDpfold_pbacktrack (TwoDpfold_vars * vars, int d1, int d2)`

Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability. If the argument 'd1' is set to '-1', the structure will be backtracked in the distance class where all structures exceeding the maximum basepair distance to either of the references reside.

#### Precondition:

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

#### See also:

[TwoDpfoldList\(\)](#)

**Parameters:**

- ← *vars* the datastructure containing all necessary folding attributes and matrices
- ← *d1* the distance to reference1 (may be -1)
- ← *d2* the distance to reference2

**Returns:**

A sampled secondary structure in dot-bracket notation

**9.31.2.2 char\* TwoDpfold\_pbacktrack5 (TwoDpfold\_vars \* vars, int d1, int d2, unsigned int length)**

Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability. This function does essentially the same as `TwoDpfold_pbacktrack` with the only difference that partial structures, i.e. structures beginning from the 5' end with a specified length of the sequence, are backtracked

**Note:**

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

**Precondition:**

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

**See also:**

[TwoDpfold\\_pbacktrack\(\)](#), [TwoDpfoldList\(\)](#)

**Parameters:**

- ← *vars* the datastructure containing all necessary folding attributes and matrices
- ← *d1* the distance to reference1 (may be -1)
- ← *d2* the distance to reference2
- ← *length* the length of the structure beginning from the 5' end

**Returns:**

A sampled secondary structure in dot-bracket notation



## 9.32 Compute the Density of States

Collaboration diagram for Compute the Density of States:



### Variables

- int `density_of_states` [MAXDOS+1]  
*The Density of States.*

### 9.32.1 Variable Documentation

#### 9.32.1.1 int `density_of_states`[MAXDOS+1]

The Density of States. This array contains the density of states for an RNA sequences after a call to `subopt_par()`, `subopt()` or `subopt_circ()`.

#### Precondition:

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

#### See also:

`subopt_par()`, `subopt()`, `subopt_circ()`

### **9.33 Parsing and Comparing - Functions to Manipulate Structures**

## Chapter 10

# Data Structure Documentation

### 10.1 bondT Struct Reference

Base pair.

#### 10.1.1 Detailed Description

Base pair.

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

- [/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.2 bondTEn Struct Reference

Base pair with associated energy.

### 10.2.1 Detailed Description

Base pair with associated energy.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.3 cofoldF Struct Reference

### Data Fields

- double **FOAB**  
*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*

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

- [/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.4 ConcEnt Struct Reference

### Data Fields

- double [AO](#)  
*start concentration A*
- double [BO](#)  
*start concentration B*
- double [ABc](#)  
*End concentration AB.*

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.5 constrain Struct Reference

constraints for cofolding

### 10.5.1 Detailed Description

constraints for cofolding

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.6 COORDINATE Struct Reference

this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

### 10.6.1 Detailed Description

this is a workarround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)



## 10.7 cpair Struct Reference

this datastructure is used as input parameter in functions of PS\_dot.c

### 10.7.1 Detailed Description

this datastructure is used as input parameter in functions of PS\_dot.c

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.8 duplexT Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.9 dupVar Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.10 folden Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.11 interact Struct Reference

### Data Fields

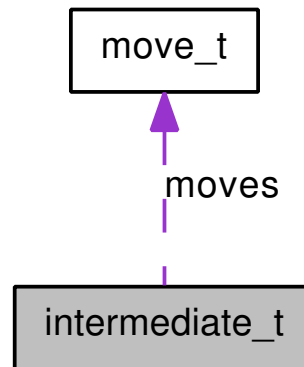
- double \* **Pi**  
*probabilities of interaction*
- double \* **Gi**  
*free energies of interaction*
- double **Gikjl**  
*full free energy for interaction between [k,i]  $k < i$  in longer seq and [j,l]  $j < l$  in shorter seq*
- double **Gikjl\_wo**  
*Gikjl without contributions for prob\_unpaired.*
- 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*

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.12 intermediate\_t Struct Reference

Collaboration diagram for intermediate\_t:



### Data Fields

- short \* `pt`  
*pair table*
- int `Sen`  
*saddle energy so far*
- int `curr_en`  
*current energy*
- `move_t` \* `moves`  
*remaining moves to target*

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.13 INTERVAL Struct Reference

Sequence interval stack element used in subopt.c.

### 10.13.1 Detailed Description

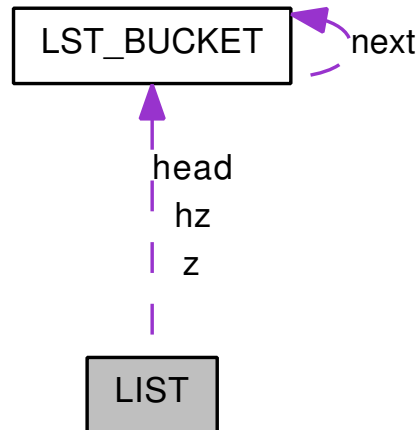
Sequence interval stack element used in subopt.c.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.14 LIST Struct Reference

Collaboration diagram for LIST:



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

- /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/lib/list.h



## 10.15 LST\_BUCKET Struct Reference

Collaboration diagram for LST\_BUCKET:



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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/lib/list.h](#)

## 10.16 model\_detailsT Struct Reference

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

### Data Fields

- int [dangles](#)  
*Specifies the dangle model used in any energy evaluation (0,1,2 or 3).*
- int [special\\_hp](#)  
*Include special hairpin contributions for tri, tetra and hexaloops.*
- int [noLP](#)  
*Only consider canonical structures, i.e. no 'lonely' base pairs.*
- int [noGU](#)  
*Do not allow GU pairs.*
- int [noGUclosure](#)  
*Do not allow loops to be closed by GU pair.*
- int [logML](#)  
*Use logarithmic scaling for multi loops.*
- int [circ](#)  
*Assume molecule to be circular.*
- int [gquad](#)  
*Include G-quadruplexes in structure prediction.*

### 10.16.1 Detailed Description

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

### 10.16.2 Field Documentation

#### 10.16.2.1 int model\_detailsT::dangles

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

**Note:**

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

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.17 move\_t Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.18 PAIR Struct Reference

Base pair data structure used in subopt.c.

### 10.18.1 Detailed Description

Base pair data structure used in subopt.c.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.19 pair\_info Struct Reference

A base pair info structure.

### Data Fields

- unsigned **i**  
*nucleotide position i*
- unsigned **j**  
*nucleotide position j*
- float **p**  
*Probability.*
- float **ent**  
*Pseudo entropy for  $p(i, j) = S_i + S_j - p_{ij} * \ln(p_{ij})$ .*
- short **bp** [8]  
*Frequencies of pair\_types.*
- char **comp**  
*1 iff pair is in mfe structure*

### 10.19.1 Detailed Description

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

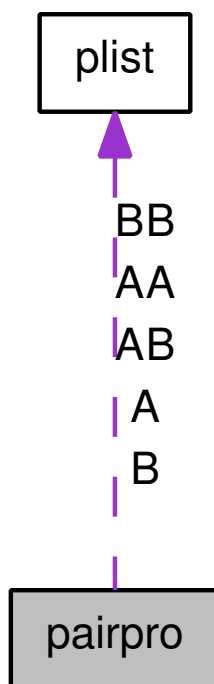
- its probability 'p'
- an entropy-like measure for its well-definedness 'ent'
- the frequency of each type of pair in 'bp[]' + 'bp[0]' contains the number of non-compatible sequences + 'bp[1]' the number of CG pairs, etc.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.20 pairpro Struct Reference

Collaboration diagram for pairpro:

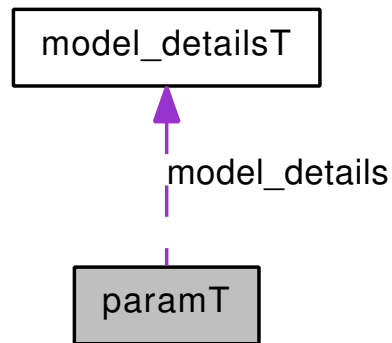


The documentation for this struct was generated from the following file:

- [/homes/fimmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.21 paramT Struct Reference

The datastructure that contains temperature scaled energy parameters. Collaboration diagram for paramT:



### Data Fields

- double [temperature](#)  
*Temperature used for loop contribution scaling.*
- [model\\_detailsT model\\_details](#)  
*Model details to be used in the recursions.*

### 10.21.1 Detailed Description

The datastructure that contains temperature scaled energy parameters.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)



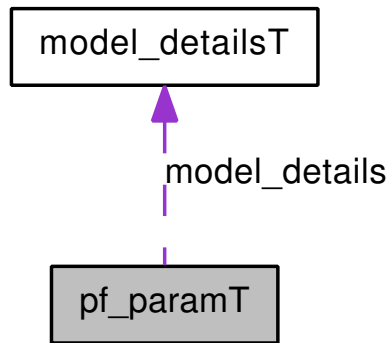
## 10.22 path\_t Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.23 pf\_paramT Struct Reference

The datastructure that contains temperature scaled Boltzmann weights of the energy parameters. Collaboration diagram for pf\_paramT:



### Data Fields

- double `pf_scale`  
*Scaling factor to avoid over-/underflows.*
- double `temperature`  
*Temperature used for loop contribution scaling.*
- double `alpha`  
*Scaling factor for the thermodynamic temperature.*
- `model_detailsT model_details`  
*Model details to be used in the recursions.*

### 10.23.1 Detailed Description

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

### 10.23.2 Field Documentation

#### 10.23.2.1 double pf\_paramT::alpha

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

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data_structures.h)

## 10.24 plist Struct Reference

this datastructure is used as input parameter in functions of [PS\\_dot.h](#) and others

### 10.24.1 Detailed Description

this datastructure is used as input parameter in functions of [PS\\_dot.h](#) and others

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.25 Postorder\_list Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist\\_vars.h](#)

## 10.26 pu\_contrib Struct Reference

contributions to p\_u

### Data Fields

- double \*\* [H](#)  
*hairpin loops*
- double \*\* [I](#)  
*interior loops*
- double \*\* [M](#)  
*multi loops*
- double \*\* [E](#)  
*exterior loop*
- int [length](#)  
*length of the input sequence*
- int [w](#)  
*longest unpaired region*

### 10.26.1 Detailed Description

contributions to p\_u

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.27 pu\_out Struct Reference

Collection of all free\_energy of beeing unpaired values for output.

### Data Fields

- int `len`  
*sequence length*
- int `u_vals`  
*number of different -u values*
- int `contribs`  
*[-c "SHIME"]*
- char \*\* `header`  
*header line*
- double \*\* `u_values`  
*(the -u values \* [-c "SHIME"]) \* seq len*

### 10.27.1 Detailed Description

Collection of all free\_energy of beeing unpaired values for output.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.28 sect Struct Reference

Stack of partial structures for backtracking.

### 10.28.1 Detailed Description

Stack of partial structures for backtracking.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)



## 10.29 snoopT Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.30 SOLUTION Struct Reference

Solution element from subopt.c.

### Data Fields

- float [energy](#)  
*Free Energy of structure in kcal/mol.*
- char \* [structure](#)  
*Structure in dot-bracket notation.*

### 10.30.1 Detailed Description

Solution element from subopt.c.

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## **10.31 struct\_en Struct Reference**

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

- /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/move\_set.h

## 10.32 svm\_model Struct Reference

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

- /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/svm\_utils.h

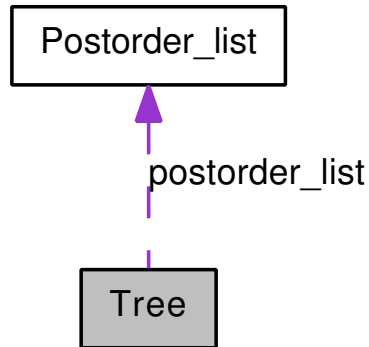
## 10.33 swString Struct Reference

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist\\_vars.h](#)

## 10.34 Tree Struct Reference

Collaboration diagram for Tree:



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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist\\_vars.h](#)

## 10.35 TwoDfold\_solution Struct Reference

Solution element returned from TwoDfoldList.

### Data Fields

- int [k](#)  
*Distance to first reference.*
- int [l](#)  
*Distance to second reference.*
- float [en](#)  
*Free energy in kcal/mol.*
- char \* [s](#)  
*MFE representative structure in dot-bracket notation.*

### 10.35.1 Detailed Description

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

A value of [INF](#) in k denotes the end of a list

#### See also:

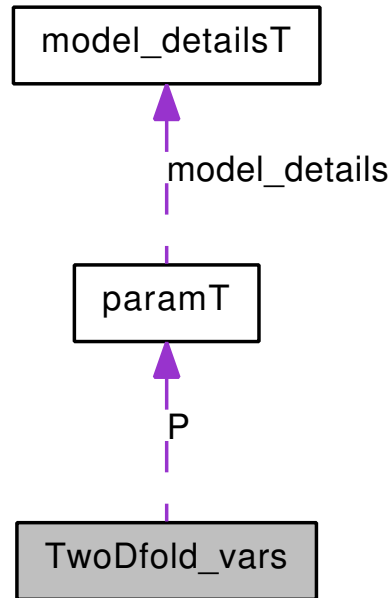
[TwoDfoldList\(\)](#)

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.36 TwoDfold\_vars Struct Reference

Variables compound for 2Dfold MFE folding. Collaboration diagram for TwoDfold\_vars:



### Data Fields

- `paramT * P`  
*Precomputed energy parameters and model details.*
- `int do_backtrack`  
*Flag whether to do backtracing of the structure(s) or not.*
- `char * ptype`  
*Precomputed array of pair types.*
- `char * sequence`  
*The input sequence.*
- `short * S1`  
*The input sequences in numeric form.*
- `unsigned int maxD1`  
*Maximum allowed base pair distance to first reference.*
- `unsigned int maxD2`



*Maximum allowed base pair distance to second reference.*

- unsigned int \* [mm1](#)  
*Maximum matching matrix, reference struct 1 disallowed.*
- unsigned int \* [mm2](#)  
*Maximum matching matrix, reference struct 2 disallowed.*
- int \* [my\\_iindx](#)  
*Index for moving in quadratic distancy dimensions.*
- unsigned int \* [referenceBPs1](#)  
*Matrix containing number of basepairs of reference structure1 in interval [i,j].*
- unsigned int \* [referenceBPs2](#)  
*Matrix containing number of basepairs of reference structure2 in interval [i,j].*
- unsigned int \* [bpdist](#)  
*Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].*

### 10.36.1 Detailed Description

Variables compound for 2Dfold MFE folding.

**See also:**

[get\\_TwoDfold\\_variables\(\)](#), [destroy\\_TwoDfold\\_variables\(\)](#), [TwoDfoldList\(\)](#)

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.37 TwoDpfold\_solution Struct Reference

Solution element returned from TwoDpfoldList.

### Data Fields

- int [k](#)  
*Distance to first reference.*
- int [l](#)  
*Distance to second reference.*
- double [q](#)  
*partition function*

### 10.37.1 Detailed Description

Solution element returned from TwoDpfoldList. This element contains the partition function for the appropriate kappa (k), lambda (l) neighborhood. The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'q' of type FLT\_OR\_DBL.

A value of [INF](#) in k denotes the end of a list.

#### See also:

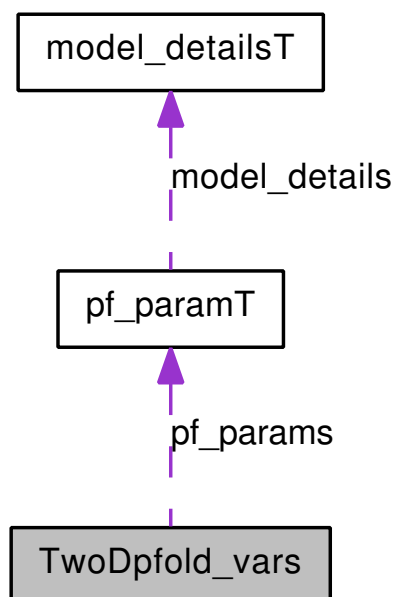
[TwoDpfoldList\(\)](#)

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

- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

## 10.38 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.*

### 10.38.1 Detailed Description

Variables compound for 2Dfold partition function folding.

**See also:**

[get\\_TwoDpfold\\_variables\(\)](#), [get\\_TwoDpfold\\_variables\\_from\\_MFE\(\)](#), [destroy\\_TwoDpfold\\_variables\(\)](#), [TwoDpfoldList\(\)](#)

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

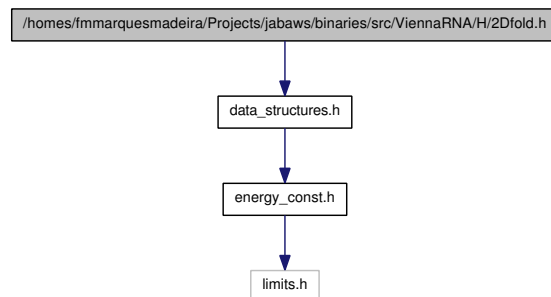
- [/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_structures.h](#)

# Chapter 11

## File Documentation

### 11.1 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/2D File Reference

Include dependency graph for 2Dfold.h:



#### Functions

- `TwoDfold_vars` \* `get_TwoDfold_variables` (`const char *seq`, `const char *structure1`, `const char *structure2`, `int circ`)  
*Get a structure of type `TwoDfold_vars` prefilled with current global settings.*
- `void destroy_TwoDfold_variables` (`TwoDfold_vars *our_variables`)  
*Destroy a `TwoDfold_vars` datastructure without memory loss.*
- `TwoDfold_solution` \* `TwoDfoldList` (`TwoDfold_vars *vars`, `int distance1`, `int distance2`)  
*Compute MFE's and representative for distance partitioning.*

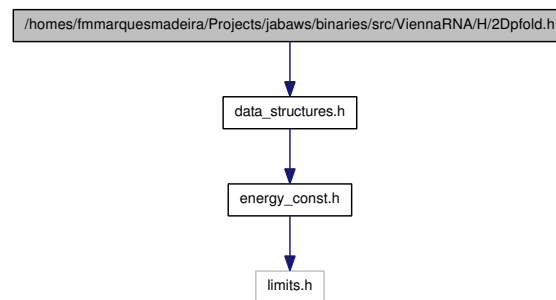
- char \* [TwoDfold\\_backtrack\\_f5](#) (unsigned int j, int k, int l, [TwoDfold\\_vars](#) \*vars)

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

### 11.1.1 Detailed Description

## File Reference

Include dependency graph for 2Dpfold.h:



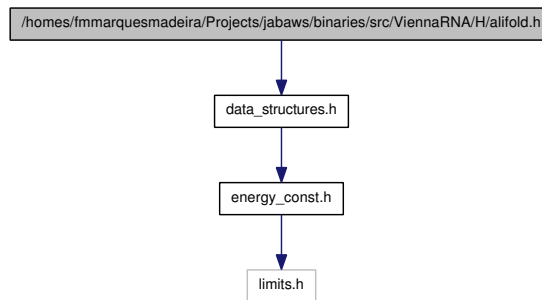
## Functions

- `TwoDpfold_vars * get_TwoDpfold_variables` (`const char *seq`, `const char *structure1`, `char *structure2`, `int circ`)  
*Get a datastructure containing all necessary attributes and global folding switches.*
- `TwoDpfold_vars * get_TwoDpfold_variables_from_MFE` (`TwoDfold_vars *mfe_vars`)  
*Get the datastructure containing all necessary attributes and global folding switches from a pre-filled mfe-datastructure.*
- `void destroy_TwoDpfold_variables` (`TwoDpfold_vars *vars`)  
*Free all memory occupied by a `TwoDpfold_vars` datastructure.*
- `TwoDpfold_solution * TwoDpfoldList` (`TwoDpfold_vars *vars`, `int maxDistance1`, `int maxDistance2`)  
*Compute the partition function for all distance classes.*
- `char * TwoDpfold_pbacktrack` (`TwoDpfold_vars *vars`, `int d1`, `int d2`)  
*Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- `char * TwoDpfold_pbacktrack5` (`TwoDpfold_vars *vars`, `int d1`, `int d2`, `unsigned int length`)  
*Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.*

### 11.2.1 Detailed Description

## 11.3 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/alifold.h File Reference

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments Include dependency graph for alifold.h:



### Functions

- void [update\\_alifold\\_params](#) (void)  
*Update the energy parameters for alifold function.*
- float [alifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according consensus structure of an alignment of sequences.*
- float [circularifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.*
- void [free\\_alifold\\_arrays](#) (void)  
*Free the memory occupied by MFE alifold functions.*
- int [get\\_mpi](#) (char \*Aseq[ ], int n\_seq, int length, int \*mini)  
*Get the mean pairwise identity in steps from ?to?(ident).*
- float \*\* [readribosum](#) (char \*name)  
*Read a ribosum or other user-defined scoring matrix.*
- float [energy\\_of\\_alistruct](#) (const char \*\*sequences, const char \*structure, int n\_seq, float \*energy)  
*Calculate the free energy of a consensus structure given a set of aligned sequences.*
- void [encode\\_ali\\_sequence](#) (const char \*sequence, short \*S, short \*s5, short \*s3, char \*ss, unsigned short \*as, int circ)  
*Get arrays with encoded sequence of the alignment.*



## 11.3

- void `alloc_sequence_arrays` (const char \*\*sequences, short \*\*\*S, short \*\*\*S5, short \*\*\*S3, unsigned short \*\*\*a2s, char \*\*\*Ss, int circ)  
*Allocate memory for sequence array used to deal with aligned sequences.*
- void `free_sequence_arrays` (unsigned int n\_seq, short \*\*\*S, short \*\*\*S5, short \*\*\*S3, unsigned short \*\*\*a2s, char \*\*\*Ss)  
*Free the memory of the sequence arrays used to deal with aligned sequences.*
- float `alipf_fold_par` (const char \*\*sequences, char \*structure, `plist **pl`, `pf_paramT *parameters`, int calculate\_bppm, int is\_constrained, int is\_circular)
- float `alipf_fold` (const char \*\*sequences, char \*structure, `plist **pl`)  
*The partition function version of `alifold()` works in analogy to `pf_fold()`. Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of `pair_info` structs. The list is terminated by the first entry with `pi.i = 0`.*
- float `alipf_circ_fold` (const char \*\*sequences, char \*structure, `plist **pl`)
- `FLT_OR_DBL * export_ali_bppm` (void)  
*Get a pointer to the base pair probability array.*
- char \* `alipbacktrack` (double \*prob)  
*Sample a consensus secondary structure from the Boltzmann ensemble according its probability*  
.
- int `get_alipf_arrays` (short \*\*\*S\_p, short \*\*\*S5\_p, short \*\*\*S3\_p, unsigned short \*\*\*a2s\_p, char \*\*\*Ss\_p, `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.*

## 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.*

### 11.3.1 Detailed Description

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

## 11.3.2 Function Documentation

### 11.3.2.1 void update\_alifold\_params (void)

Update the energy parameters for alifold function. Call this to recalculate the pair matrix and energy parameters after a change in folding parameters like [temperature](#)

## 11.4

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/cofold.h

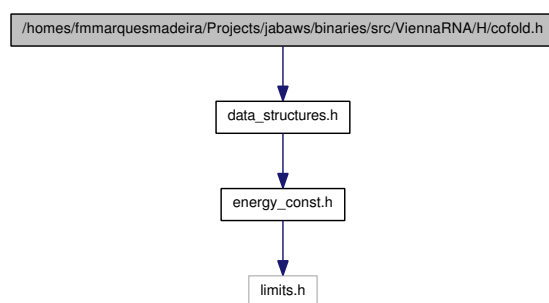
File Reference

11.4 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/co

159

### File Reference

MFE version of cofolding routines. Include dependency graph for cofold.h:



### Functions

- float `cofold` (const char \*sequence, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*
- float `cofold_par` (const char \*string, char \*structure, `paramT` \*parameters, int is\_constrained)  
*Compute the minimum free energy of two interacting RNA molecules.*
- void `free_co_arrays` (void)  
*Free memory occupied by `cofold()`.*
- void `update_cofold_params` (void)  
*Recalculate parameters.*
- void `export_cofold_arrays_gq` (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function `cofold` (with `gquadruplex` support).*
- void `export_cofold_arrays` (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function `cofold`.*
- `SOLUTION` \* `zuckersubopt` (const char \*string)  
*Compute Zuker type suboptimal structures.*
- `SOLUTION` \* `zuckersubopt_par` (const char \*string, `paramT` \*parameters)  
*Compute Zuker type suboptimal structures.*
- void `get_monomere_mfes` (float \*e1, float \*e2)

*get\_monomer\_free\_energies*

- void [initialize\\_cofold](#) (int length)

### 11.4.1 Detailed Description

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

### 11.4.2 Function Documentation

#### 11.4.2.1 void [get\\_monomere\\_mfes](#) (float \* *e1*, float \* *e2*)

[get\\_monomer\\_free\\_energies](#) Export monomer free energies out of cofold arrays

**Parameters:**

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

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

#### 11.4.2.2 void [initialize\\_cofold](#) (int *length*)

allocate arrays for folding

**Deprecated**

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

## 11.5

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/convert\_ -

epars.h File  
Reference

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/co

### epars.h File Reference

Functions and definitions for energy parameter file format conversion.

#### Defines

- #define [VRNA\\_CONVERT\\_OUTPUT\\_ALL](#) 1U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_HP](#) 2U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_STACK](#) 4U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_HP](#) 8U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT](#) 16U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT\\_1N](#) 32U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT\\_23](#) 64U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_MULTI](#) 128U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_EXT](#) 256U
- #define [VRNA\\_CONVERT\\_OUTPUT\\_DANGLES](#) 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

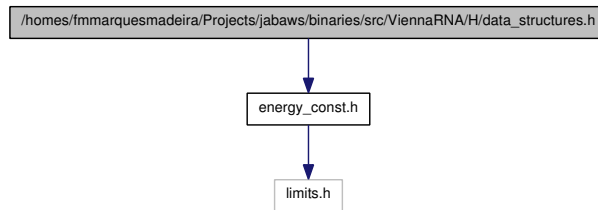
- void [convert\\_parameter\\_file](#) (const char \*iname, const char \*oname, unsigned int options)

#### 11.5.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

## 11.6 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/structures.h File Reference

All datastructures and typedefs shared among the Vienna RNA Package can be found here. Include dependency graph for data\_structures.h:



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



### Data Structures

- struct [plist](#)  
*this datastructure is used as input parameter in functions of [PS\\_dot.h](#) and others*
- struct [cpair](#)  
*this datastructure is used as input parameter in functions of [PS\\_dot.c](#)*
- struct [COORDINATE](#)  
*this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)*
- struct [sect](#)  
*Stack of partial structures for backtracking.*
- struct [bondT](#)  
*Base pair.*
- struct [bondTE](#)  
*Base pair with associated energy.*
- struct [model\\_detailsT](#)  
*The data structure that contains the complete model details used throughout the calculations.*
- struct [paramT](#)

## 11.6

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\_structures.h File

**Reference** *The datastructure that contains temperature scaled energy parameters.*

163

- struct [pf\\_paramT](#)  
*The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.*
- struct [PAIR](#)  
*Base pair data structure used in subopt.c.*
- struct [INTERVAL](#)  
*Sequence interval stack element used in subopt.c.*
- struct [SOLUTION](#)  
*Solution element from subopt.c.*
- struct [cofoldF](#)
- struct [ConcEnt](#)
- struct [pairpro](#)
- struct [pair\\_info](#)  
*A base pair info structure.*
- struct [move\\_t](#)
- struct [intermediate\\_t](#)
- struct [path\\_t](#)
- struct [pu\\_contrib](#)  
*contributions to p\_u*
- struct [interact](#)
- struct [pu\\_out](#)  
*Collection of all free\_energy of being unpaired values for output.*
- struct [constrain](#)  
*constraints for cofolding*
- struct [duplexT](#)
- struct [folden](#)
- struct [snoopT](#)
- struct [dupVar](#)
- struct [TwoDfold\\_solution](#)  
*Solution element returned from TwoDfoldList.*
- struct [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding.*
- struct [TwoDpfold\\_solution](#)  
*Solution element returned from TwoDpfoldList.*

- struct [TwoDpfold\\_vars](#)  
*Variables compound for 2Dfold partition function folding.*

## Defines

- #define [MAXALPHA](#) 20  
*Maximal length of alphabet.*
- #define [MAXDOS](#) 1000  
*Maximum density of states discretization for subopt.*

### 11.6.1 Detailed Description

All datastructures and typedefs shared among the Vienna RNA Package can be found [here](#).



## 11.7

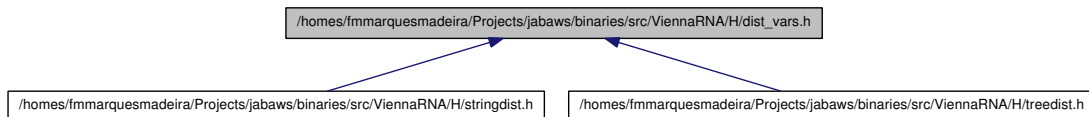
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist\_

vars.h File  
Reference

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dis

### vars.h File Reference

Global variables for Distance-Package. This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [Postorder\\_list](#)
- struct [Tree](#)
- struct [swString](#)

## Variables

- int [edit\\_backtrack](#)  
*Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.*
- char \* [aligned\\_line](#) [4]  
*Contains the two aligned structures after a call to one of the distance functions with [edit\\_backtrack](#) set to 1.*
- int [cost\\_matrix](#)  
*Specify the cost matrix to be used for distance calculations.*

### 11.7.1 Detailed Description

Global variables for Distance-Package.

### 11.7.2 Variable Documentation

#### 11.7.2.1 int edit\_backtrack

Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance. set to 1 if you want backtracking

**11.7.2.2 int cost\_matrix**

Specify the cost matrix to be used for distance calculations. if 0, use the default cost matrix (upper matrix in example), otherwise use Shapiro's costs (lower matrix).

## 11.8

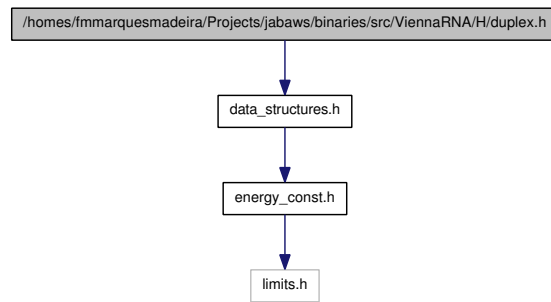
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/duplex.h

File Reference

11.8 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/du<sup>167</sup>

### File Reference

Duplex folding function declarations... Include dependency graph for duplex.h:



### 11.8.1 Detailed Description

Duplex folding function declarations...

## **11.9 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRN cost.h File Reference**

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

### **11.9.1 Detailed Description**

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

## 11.10

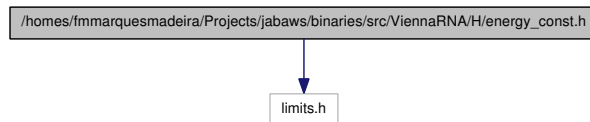
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/energy\_ -

const.h File  
Reference

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/e

### const.h File Reference

Include dependency graph for energy\_const.h:



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



## Defines

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

### 11.10.1 Detailed Description

energy constants

### 11.10.2 Define Documentation

#### 11.10.2.1 #define GASCONST 1.98717

The gas constant

#### 11.10.2.2 #define K0 273.15

0 deg Celsius in Kelvin

#### 11.10.2.3 #define INF 10000000

Infinity as used in minimization routines

**11.10.2.4 #define FORBIDDEN 9999**

forbidden

**11.10.2.5 #define BONUS 10000**

bonus contribution

**11.10.2.6 #define NBPAIRS 7**

The number of distinguishable base pairs

**11.10.2.7 #define TURN 3**

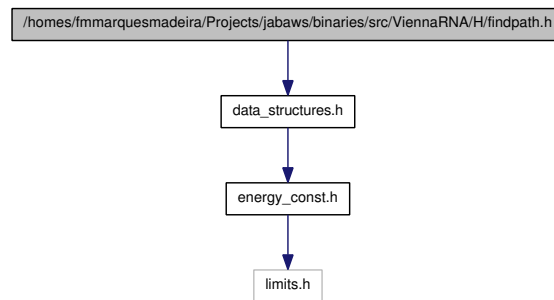
The minimum loop length

**11.10.2.8 #define MAXLOOP 30**

The maximum loop length

## File Reference

Compute direct refolding paths between two secondary structures. Include dependency graph for findpath.h:



## Functions

- int [find\\_saddle](#) (const char \*seq, const char \*struc1, const char \*struc2, int max)  
*Find energy of a saddle point between 2 structures (serch only direct path).*
- path\_t \* [get\\_path](#) (const char \*seq, const char \*s1, const char \*s2, int maxkeep)  
*Find refolding path between 2 structures (serch only direct path).*
- void [free\\_path](#) (path\_t \*path)  
*Free memory allocated by [get\\_path\(\)](#) function.*

### 11.11.1 Detailed Description

Compute direct refolding paths between two secondary structures.

### 11.11.2 Function Documentation

**11.11.2.1 int [find\\_saddle](#) (const char \* seq, const char \* struc1, const char \* struc2, int max)**

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

#### Parameters:

*seq* RNA sequence

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

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

*max* integer how many structures are being kept during the search

**Returns:**

the saddle energy in 10cal/mol

**11.11.2.2 path\_t\* get\_path (const char \* seq, const char \* s1, const char \* s2, int maxkeep)**

Find refolding path 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

*maxkeep* integer how many structures are being kept during the search

**Returns:**

direct refolding path between two structures

**11.11.2.3 void free\_path (path\_t \* path)**

Free memory allocated by [get\\_path\(\)](#) function.

**Parameters:**

*path* pointer to memory to be freed



## 11.12

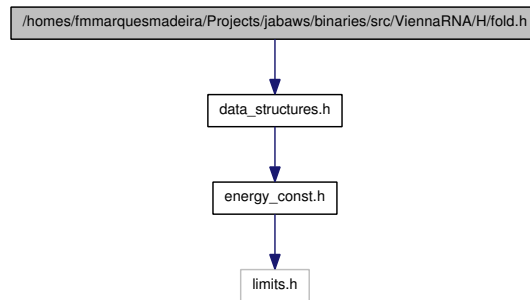
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h

File Reference

11.12 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h

### File Reference

MFE calculations and energy evaluations for single RNA sequences. Include dependency graph for fold.h:



### Functions

- float `fold_par` (const char \*sequence, char \*structure, paramT \*parameters, int is\_constrained, int is\_circular)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float `fold` (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float `circfold` (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.*
- float `energy_of_structure` (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded RNA using global model detail settings.*
- float `energy_of_struct_par` (const char \*string, const char \*structure, paramT \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float `energy_of_circ_structure` (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- float `energy_of_circ_struct_par` (const char \*string, const char \*structure, paramT \*parameters, int verbosity\_level)

*Calculate the free energy of an already folded circular RNA.*

- int [energy\\_of\\_structure\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)

*Calculate the free energy of an already folded RNA.*

- int [energy\\_of\\_struct\\_pt\\_par](#) (const char \*string, short \*ptable, short \*s, short \*s1, [paramT](#) \*parameters, int verbosity\_level)

*Calculate the free energy of an already folded RNA.*

- void [free\\_arrays](#) (void)

*Free arrays for mfe folding.*

- void [parenthesis\\_structure](#) (char \*structure, [bondT](#) \*bp, int length)

*Create a dot-bracket/parenthesis structure from backtracking stack.*

- void [parenthesis\\_zuker](#) (char \*structure, [bondT](#) \*bp, int length)

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

- void [update\\_fold\\_params](#) (void)

*Recalculate energy parameters.*

- float [energy\\_of\\_move](#) (const char \*string, const char \*structure, int m1, int m2)

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

- int [energy\\_of\\_move\\_pt](#) (short \*pt, short \*s, short \*s1, int m1, int m2)

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

- int [loop\\_energy](#) (short \*ptable, short \*s, short \*s1, int i)

*Calculate energy of a loop.*

- void [assign\\_plist\\_from\\_db](#) ([plist](#) \*\*pl, const char \*struc, float pr)

*Create a [plist](#) from a dot-bracket string.*

- int [LoopEnergy](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1)

- int [HairpinE](#) (int size, int type, int si1, int sj1, const char \*string)

- void [initialize\\_fold](#) (int length)

- float [energy\\_of\\_struct](#) (const char \*string, const char \*structure)

- int [energy\\_of\\_struct\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1)

- float [energy\\_of\\_circ\\_struct](#) (const char \*string, const char \*structure)

---

**Variables**

- int `logML`  
*if nonzero use logarithmic ML energy in energy\_of\_struct*
- int `uniq_ML`  
*do ML decomposition uniquely (for subopt)*
- int `cut_point`  
*set to first pos of second seq for cofolding*
- int `eos_debug`  
*verbose info from energy\_of\_struct*

### 11.12.1 Detailed Description

MFE calculations and energy evaluations for single RNA sequences. This file includes (almost) all function declarations within the RNAlib that are related to MFE folding...

### 11.12.2 Function Documentation

#### 11.12.2.1 void `parenthesis_structure` (char \* *structure*, bondT \* *bp*, int *length*)

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

**Note:**

This function is threadsafe

#### 11.12.2.2 void `parenthesis_zuker` (char \* *structure*, bondT \* *bp*, int *length*)

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

**Note:**

This function is threadsafe

#### 11.12.2.3 float `energy_of_move` (const char \* *string*, const char \* *structure*, int *m1*, int *m2*)

Calculate energy of a move (closing or opening of a base pair). If the parameters *m1* and *m2* are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

**See also:**

[make\\_pair\\_table\(\)](#), [energy\\_of\\_move\(\)](#)

**Parameters:**

*string* RNA sequence  
*structure* secondary structure in dot-bracket notation  
*m1* first coordinate of base pair  
*m2* second coordinate of base pair

**Returns:**

energy change of the move in kcal/mol

**11.12.2.4 int energy\_of\_move\_pt (short \* pt, short \* s, short \* s1, int m1, int m2)**

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

**See also:**

[make\\_pair\\_table\(\)](#), [energy\\_of\\_move\(\)](#)

**Parameters:**

*pt* the pair table of the secondary structure  
*s* encoded RNA sequence  
*s1* encoded RNA sequence  
*m1* first coordinate of base pair  
*m2* second coordinate of base pair

**Returns:**

energy change of the move in 10cal/mol

**11.12.2.5 int loop\_energy (short \* ptable, short \* s, short \* s1, int i)**

Calculate energy of a loop.

**Parameters:**

*ptable* the pair table of the secondary structure  
*s* encoded RNA sequence  
*s1* encoded RNA sequence

## 11.12

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h

### File Reference

177

---

*i* position of covering base pair

#### Returns:

free energy of the loop in 10cal/mol

#### 11.12.2.6 void assign\_plist\_from\_db (plist \*\* *pl*, const char \* *struc*, float *pr*)

Create a [plist](#) from a dot-bracket string. The dot-bracket string is parsed and for each base pair an entry in the [plist](#) is created. The probability of each pair in the list is set by a function parameter.

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

This function is threadsafe

#### Parameters:

*pl* A pointer to the [plist](#) that is to be created

*struc* The secondary structure in dot-bracket notation

*pr* The probability for each base pair

#### 11.12.2.7 int LoopEnergy (int *n1*, int *n2*, int *type*, int *type\_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*)

#### Deprecated

{This function is deprecated and will be removed soon. Use [E\\_IntLoop\(\)](#) instead!}

#### 11.12.2.8 int HairpinE (int *size*, int *type*, int *si1*, int *sj1*, const char \* *string*)

#### Deprecated

{This function is deprecated and will be removed soon. Use [E\\_Hairpin\(\)](#) instead!}

#### 11.12.2.9 void initialize\_fold (int *length*)

Allocate arrays for folding

#### Deprecated

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

### 11.12.2.10 float `energy_of_struct` (const char \* *string*, const char \* *structure*)

Calculate the free energy of an already folded RNA

**Note:**

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

**Deprecated**

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

**See also:**

[energy\\_of\\_structure](#), [energy\\_of\\_circ\\_struct\(\)](#), [energy\\_of\\_struct\\_pt\(\)](#)

**Parameters:**

*string* RNA sequence

*structure* secondary structure in dot-bracket notation

**Returns:**

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

### 11.12.2.11 int `energy_of_struct_pt` (const char \* *string*, short \* *ptable*, short \* *s*, short \* *sI*)

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

## 11.12

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h

### File Reference

179

---

*s* encoded RNA sequence

*sI* encoded RNA sequence

#### Returns:

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

#### 11.12.2.12 float energy\_of\_circ\_struct (const char \* *string*, const char \* *structure*)

Calculate the free energy of an already folded circular RNA

#### Note:

This function is not entirely threadsafe! Depending on the state of the global variable [eos\\_debug](#) it prints energy information to stdout or not...

#### Deprecated

This function is deprecated and should not be used in future programs Use [energy\\_of\\_circ\\_structure\(\)](#) instead!

#### See also:

[energy\\_of\\_circ\\_structure\(\)](#), [energy\\_of\\_struct\(\)](#), [energy\\_of\\_struct\\_pt\(\)](#)

#### Parameters:

*string* RNA sequence

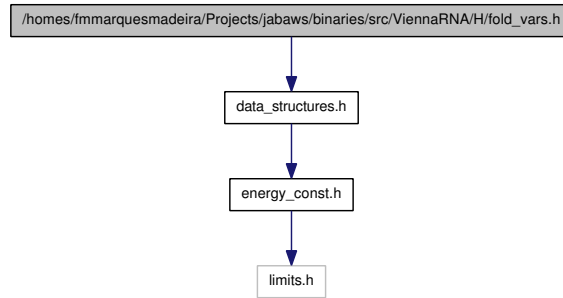
*structure* secondary structure in dot-bracket notation

#### Returns:

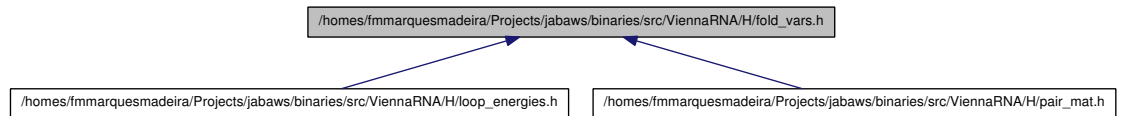
the free energy of the input structure given the input sequence in kcal/mol

## 11.13 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold\_vars.h File Reference

Here all all declarations of the global variables used throughout RNAlib. Include dependency graph for fold\_vars.h:



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



### Functions

- void `set_model_details` (`model_detailsT *md`)  
*Set default model details.*

### Variables

- int `fold_constrained`  
*Global switch to activate/deactivate folding with structure constraints.*
- int `noLonelyPairs`  
*Global switch to avoid/allow helices of length 1.*
- int `dangles`  
*Switch the energy model for dangling end contributions (0, 1, 2, 3).*
- int `noGU`  
*Global switch to forbid/allow GU base pairs at all.*



### 11.13

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold\_vars.h File

Reference: [int\\_no\\_closingGU](#)

181

---

*GU allowed only inside stacks if set to 1.*

- int [tetra\\_loop](#)  
*Include special stabilizing energies for some tri-, tetra- and hexa-loops;.*
- int [energy\\_set](#)  
*0 = BP; 1=any mit GC; 2=any mit AU-parameter*
- int [circ](#)  
*backward compatibility variable.. this does not effect anything*
- int [csv](#)  
*generate comma seperated output*
- int [oldAliEn](#)
- int [ribo](#)
- char \* [RibosumFile](#)
- char \* [nonstandards](#)  
*contains allowed non standard base pairs*
- double [temperature](#)  
*Rescale energy parameters to a temperature in degC.*
- int [james\\_rule](#)
- int [logML](#)
- int [cut\\_point](#)  
*Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.*
- [bondT](#) \* [base\\_pair](#)  
*Contains a list of base pairs after a call to [fold\(\)](#).*
- double \* [pr](#)  
*A pointer to the base pair probability matrix.*
- int \* [iindx](#)  
*index array to move through pr.*
- double [pf\\_scale](#)  
*A scaling factor used by [pf\\_fold\(\)](#) to avoid overflows.*
- int [do\\_backtrack](#)  
*do backtracking, i.e. compute secondary structures or base pair probabilities*
- char [backtrack\\_type](#)

A backtrack array marker for *inverse\_fold()*.

- int `gquad`  
Allow G-quadruplex formation.

### 11.13.1 Detailed Description

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

### 11.13.2 Function Documentation

#### 11.13.2.1 void `set_model_details(model_detailsT * md)`

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

**See also:**

#### Parameters:

*md* A pointer to the data structure that shall be initialized

### 11.13.3 Variable Documentation

#### 11.13.3.1 int `noLonelyPairs`

Global switch to avoid/allow helices of length 1. Disallow all pairs which can only occur as lonely pairs (i.e. as helix of length 1). This avoids lonely base pairs in the predicted structures in most cases.

#### 11.13.3.2 int `dangles`

Switch the energy model for dangling end contributions (0, 1, 2, 3). If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (`dangles = 1`) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm `pf_fold()` these checks are neglected. If `dangles` is set to 2, all folding routines will follow this convention. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.

If `dangles = 3` co-axial stacking is explicitly included for adjacent helices in 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.

## 11.13

**/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold\_vars.h File**

Default is 2 in most algorithms, partition function algorithms can only handle 0 and 1

### Reference

---

#### 11.13.3.3 int tetra\_loop

Include special stabilizing energies for some tri-, tetra- and hexa-loops;. default is 1.

#### 11.13.3.4 int energy\_set

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

#### 11.13.3.5 int oldAliEn

use old alifold energies (with gaps)

#### 11.13.3.6 int ribo

use ribosum matrices

#### 11.13.3.7 char\* RibosumFile

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

#### 11.13.3.8 char\* nonstandards

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

#### 11.13.3.9 double temperature

Rescale energy parameters to a temperature in degC. Default is 37C. You have to call the update\_...\_params() functions after changing this parameter.

#### 11.13.3.10 int james\_rule

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

#### 11.13.3.11 int logML

use logarithmic multiloop energy function

### 11.13.3.12 int cut\_point

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

### 11.13.3.13 bondT\* base\_pair

Contains a list of base pairs after a call to `fold()`. `base_pair[0].i` contains the total number of pairs.

#### Deprecated

Do not use this variable anymore!

### 11.13.3.14 double\* pr

A pointer to the base pair probability matrix.

#### Deprecated

Do not use this variable anymore!

### 11.13.3.15 int\* iindx

index array to move through `pr`. The probability for base `i` and `j` to form a pair is in `pr[iindx[i]-j]`.

#### Deprecated

Do not use this variable anymore!

### 11.13.3.16 double pf\_scale

A scaling factor used by `pf_fold()` to avoid overflows. Should be set to approximately  $\exp((-F/kT)/length)$ , where  $F$  is an estimate for the ensemble free energy, for example the minimum free energy. You must call `update_pf_params()` after changing this parameter.

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

## 11.13

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold\\_vars.h](#) File

**11.13.3.17** `int do_backtrack`

**185**

---

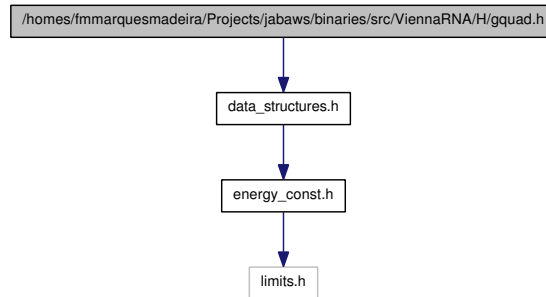
do backtracking, i.e. compute secondary structures or base pair probabilities. If 0, do not calculate pair probabilities in [pf\\_fold\(\)](#); this is about twice as fast. Default is 1.

### 11.13.3.18 `char backtrack_type`

A backtrack array marker for [inverse\\_fold\(\)](#). If set to 'C': force (1,N) to be paired, 'M' fold as if the sequence were inside a multi-loop. Otherwise ('F') the usual mfe structure is computed.

## 11.14 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA File Reference

Various functions related to G-quadruplex computations. Include dependency graph for gquad.h:



### Functions

- `int * get_gquad_matrix` (short \*S, paramT \*P)  
*Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.*
- `int parse_gquad` (const char \*struc, int \*L, int l[3])
- PRIVATE `int backtrack_GQuad_IntLoop` (int c, int i, int j, int type, short \*S, int \*ggg, int \*index, int \*p, int \*q, paramT \*P)
- PRIVATE `int backtrack_GQuad_IntLoop_L` (int c, int i, int j, int type, short \*S, int \*\*ggg, int maxdist, int \*p, int \*q, paramT \*P)

#### 11.14.1 Detailed Description

Various functions related to G-quadruplex computations.

#### 11.14.2 Function Documentation

##### 11.14.2.1 `int* get_gquad_matrix` (short \* S, paramT \* P)

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

#### See also:

`get_jindx()`, `encode_sequence()`

## 11.14

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/gquad.h

File Reference

187

### 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

#### 11.14.2.2 int parse\_gquad (const char \* *struc*, int \* *L*, int *l*[3])

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

#### 11.14.2.3 PRIVATE int backtrack\_GQuad\_IntLoop (int *c*, int *i*, int *j*, int *type*, short \* *S*, int \* *ggg*, int \* *index*, int \* *p*, int \* *q*, paramT \* *P*)

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

### Parameters:

*c* The total contribution the loop should resemble

*i* position i of enclosing pair

*j* position j of enclosing pair

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

*S* integer encoded sequence

*ggg* triangular matrix containing g-quadruplex contributions

*index* the index for accessing the triangular matrix

*p* here the 5' position of the gquad is stored

*q* here the 3' position of the gquad is stored

*P* the datastructure containing the precalculated contributions

### Returns:

1 on success, 0 if no gquad found

#### 11.14.2.4 PRIVATE int backtrack\_GQuad\_IntLoop\_L (int *c*, int *i*, int *j*, int *type*, short \* *S*, int \*\* *ggg*, int *maxdist*, int \* *p*, int \* *q*, paramT \* *P*)

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

**Parameters:**

- c* The total contribution the loop should resemble
- i* position i of enclosing pair
- j* position j of enclosing pair
- type* base pair type of enclosing pair (must be reverse type)
- S* integer encoded sequence
- ggg* triangular matrix containing g-quadruplex contributions
- p* here the 5' position of the gquad is stored
- q* here the 3' position of the gquad is stored
- P* the datastructure containing the precalculated contributions

**Returns:**

- 1 on success, 0 if no gquad found



## 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](#)

#### 11.15.1 Detailed Description

Inverse folding routines.

## 11.16 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR File Reference

Predicting local MFE structures of large sequences.

### Functions

- float [Lfold](#) (const char \*string, char \*structure, int maxdist)  
*The local analog to [fold\(\)](#).*
- float [Lfoldz](#) (const char \*string, char \*structure, int maxdist, int zsc, double min\_z)
- float [aliLfold](#) (const char \*\*strings, char \*structure, int maxdist)

### 11.16.1 Detailed Description

Predicting local MFE structures of large sequences.

## 11.17

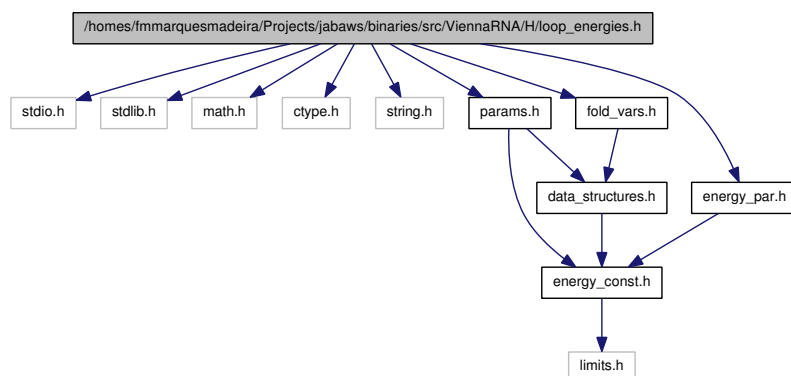
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop\_energies.h File

energies.h File

Reference /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop\_energies.h File Reference

### energies.h File Reference

Energy evaluation for MFE and partition function calculations. Include dependency graph for loop\_energies.h:



## Functions

- PRIVATE int [E\\_IntLoop](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1, [paramT](#) \*P)
- PRIVATE int [E\\_Hairpin](#) (int size, int type, int si1, int sj1, const char \*string, [paramT](#) \*P)
- PRIVATE int [E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [paramT](#) \*P)
- PRIVATE double [exp\\_E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [pf\\_paramT](#) \*P)
- PRIVATE double [exp\\_E\\_Hairpin](#) (int u, int type, short si1, short sj1, const char \*string, [pf\\_paramT](#) \*P)
- PRIVATE double [exp\\_E\\_IntLoop](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, [pf\\_paramT](#) \*P)

### 11.17.1 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  $10^{-2} * \text{kcal/mol}$

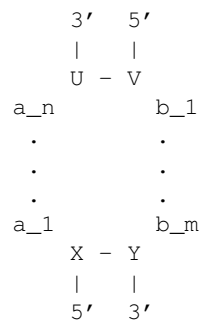
In case of computing the partition function, this file also supplies functions which return the Boltzmann weights  $e^{-\Delta G/kT}$  for a hairpin- [ [exp\\_E\\_Hairpin\(\)](#) ] or interior-loop [ [exp\\_E\\_IntLoop\(\)](#) ].

## 11.17.2 Function Documentation

**11.17.2.1 PRIVATE int E\_IntLoop (int *n1*, int *n2*, int *type*, int *type\_2*, int *si1*, int *sj1*, int *sp1*, int *sq1*, paramT \* *P*)**

### Compute the Energy of an interior-loop

This function computes the free energy  $\Delta G$  of an interior-loop with the following structure:



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 around' when evaluating the free energy of the interior-loop

#### See also:

[scale\\_parameters\(\)](#)  
paramT

#### Note:

This function is threadsafe

#### Parameters:

*n1* The size of the 'left'-loop (number of unpaired nucleotides)

## 11.17

`/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop_energies.h` File

**Reference** The size of the 'right'-loop (number of unpaired nucleotides) **193**

---

*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

**11.17.2.2 PRIVATE int E\_Hairpin (int size, int type, int si1, int sj1, const char \*string, paramT \*P)**

### Compute the Energy of a hairpin-loop

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

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

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

### Note:

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

### See also:

[scale\\_parameters\(\)](#)  
[paramT](#)

**Warning:**

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!

Energy evaluation may change due to updates in global variable "tetra\_loop"

**Parameters:**

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

*type* The pair type of the base pair closing the hairpin

*si1* The 5'-mismatching nucleotide

*sj1* The 3'-mismatching nucleotide

*string* The sequence of the loop

*P* The datastructure containing scaled energy parameters

**Returns:**

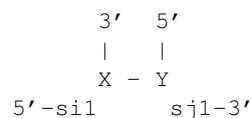
The Free energy of the Hairpin-loop in dcal/mol

### 11.17.2.3 PRIVATE int E\_Stem (int *type*, int *si1*, int *sj1*, int *extLoop*, paramT \* *P*)

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

This function computes the energy contribution of a stem that branches off a loop region. This can be the case in multiloops, when a stem branching off increases the degree of the loop but also *immediately interior base pairs* of an exterior loop contribute free energy. To switch the 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 incooperating dangling end or mismatch energies just pass a negative number, e.g. -1 to the mismatch argument.

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



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

## 11.17

`/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop_energies.h` File

**Reference** If these mismatch contributions to be incorporated, just pass a negative number **199** both, `si1` and `sj1`. In case the argument `extLoop` is 0, the returned energy contribution also includes the *internal-loop-penalty* of a multiloop stem with closing pair type.

### See also:

`E_MLstem()`  
`E_ExtLoop()`

### Note:

This function is threadsafe

### Parameters:

*type* The pair type of the first base pair un the stem  
*si1* The 5'-mismatching nucleotide  
*sj1* The 3'-mismatching nucleotide  
*extLoop* A flag that indicates whether the contribution reflects the one of an exterior loop or not  
*P* The datastructure containing scaled energy parameters

### Returns:

The Free energy of the branch off the loop in dcal/mol

**11.17.2.4 PRIVATE double exp\_E\_Stem (int type, int si1, int sj1, int extLoop, pf\_paramT \* P)**

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

This is the partition function variant of [E\\_Stem\(\)](#)

### See also:

[E\\_Stem\(\)](#)

### Note:

This function is threadsafe

### Returns:

The Boltzmann weighted energy contribution of the branch off the loop

### 11.17.2.5 PRIVATE double exp\_E\_Hairpin (int *u*, int *type*, short *si1*, short *sj1*, const char \* *string*, pf\_paramT \* *P*)

Compute Boltzmann weight  $e^{-\Delta G/kT}$  of a hairpin loop

multiply by scale[u+2]

See also:

[get\\_scaled\\_pf\\_parameters\(\)](#)  
[pf\\_paramT](#)  
[E\\_Hairpin\(\)](#)

**Warning:**

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

**Parameters:**

*u* The size of the loop (number of unpaired nucleotides)  
*type* The pair type of the base pair closing the hairpin  
*si1* The 5'-mismatching nucleotide  
*sj1* The 3'-mismatching nucleotide  
*string* The sequence of the loop  
*P* The datastructure containing scaled Boltzmann weights of the energy parameters

**Returns:**

The Boltzmann weight of the Hairpin-loop

### 11.17.2.6 PRIVATE double exp\_E\_IntLoop (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*, pf\_paramT \* *P*)

Compute Boltzmann weight  $e^{-\Delta G/kT}$  of interior loop

multiply by scale[u1+u2+2] for scaling

See also:

[get\\_scaled\\_pf\\_parameters\(\)](#)  
[pf\\_paramT](#)  
[E\\_IntLoop\(\)](#)

**Note:**

This function is threadsafe



## 11.17

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/loop\_energies.h File

Parameters:

Reference

197

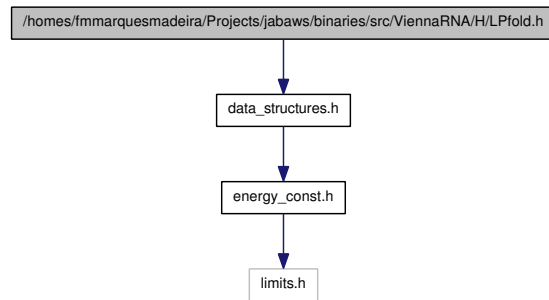
- 
- u1* The size of the 'left'-loop (number of unpaired nucleotides)
  - u2* The size of the 'right'-loop (number of unpaired nucleotides)
  - type* The pair type of the base pair closing the interior loop
  - type2* The pair type of the enclosed base pair
  - si1* The 5'-mismatching nucleotide of the 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

## 11.18 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA File Reference

Function declarations of partition function variants of the Lfold algorithm. Include dependency graph for LPfold.h:



### Functions

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

*Compute partition functions for locally stable secondary structures.*

- `plist` \* `pfl_fold_par` (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, struct `plist` \*\*dpp2, FILE \*pUfp, FILE \*spup, `pf_paramT` \*parameters)

*Compute partition functions for locally stable secondary structures.*

- void `putoutpU_prob` (double \*\*pU, int length, int ulength, FILE \*fp, int energies)

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

- void `putoutpU_prob_bin` (double \*\*pU, int length, int ulength, FILE \*fp, int energies)

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

- void `init_pf_foldLP` (int length)

### 11.18.1 Detailed Description

Function declarations of partition function variants of the Lfold algorithm.

---

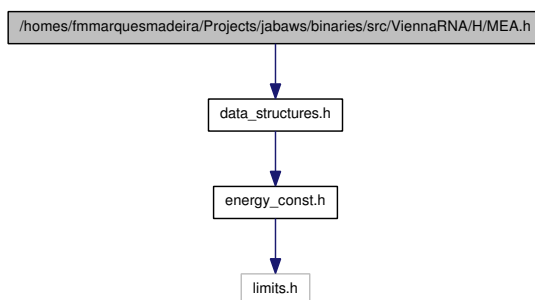
## 11.18.2 Function Documentation

### 11.18.2.1 void init\_pf\_foldLP (*int length*)

Dunno if this function was ever used by external programs linking to RNALib, but it was declared PUBLIC before. Anyway, never use this function as it will be removed soon and does nothing at all

## 11.19 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/MEA.h File Reference

Computes a MEA (maximum expected accuracy) structure. Include dependency graph for MEA.h:



### Functions

- float `MEA` (`plist *p`, `char *structure`, `double gamma`)  
*Computes a MEA (maximum expected accuracy) structure.*

#### 11.19.1 Detailed Description

Computes a MEA (maximum expected accuracy) structure.

#### 11.19.2 Function Documentation

##### 11.19.2.1 float MEA (plist \*p, char \*structure, double gamma)

Computes a MEA (maximum expected accuracy) structure. The algorithm maximizes the expected accuracy

$$A(S) = \sum_{(i,j) \in S} 2\gamma p_{ij} + \sum_{i \notin S} p_i^u$$

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.

11.20

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/mm.h

File Reference

~~11.20~~ /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/n<sup>201</sup>

## File Reference

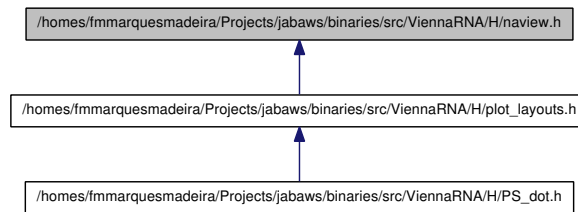
Several Maximum Matching implementations.

### 11.20.1 Detailed Description

Several Maximum Matching implementations. This file contains the declarations for several maximum matching implementations

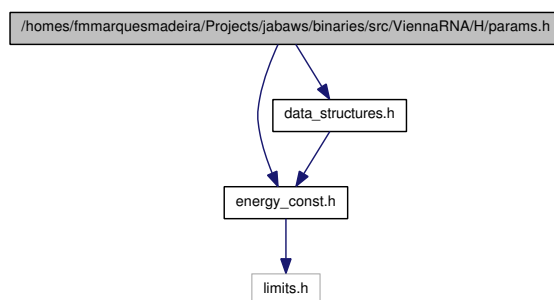
## 11.21 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA File Reference

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

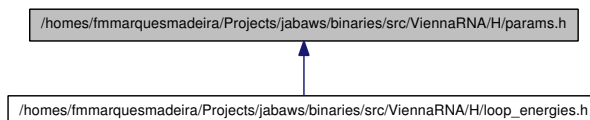


### 11.21.1 Detailed Description

Include dependency graph for params.h:



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



## Functions

- `paramT * scale_parameters` (void)  
*Get precomputed energy contributions for all the known loop types.*
- `paramT * get_scaled_parameters` (double temperature, model\_detailsT md)  
*Get precomputed energy contributions for all the known loop types.*
- `pf_paramT * get_scaled_pf_parameters` (void)
- `pf_paramT * get_boltzmann_factors` (double temperature, double betaScale, model\_detailsT md, double pf\_scale)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.*
- `pf_paramT * get_boltzmann_factor_copy` (pf\_paramT \*parameters)  
*Get a copy of already precomputed Boltzmann factors.*
- `pf_paramT * get_scaled_alipf_parameters` (unsigned int n\_seq)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant).*

- `pf_paramT * get_boltzmann_factors_ali` (unsigned int `n_seq`, double `temperature`, double `betaScale`, `model_detailsT md`, double `pf_scale`)

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

### 11.22.1 Detailed Description



## 11.23

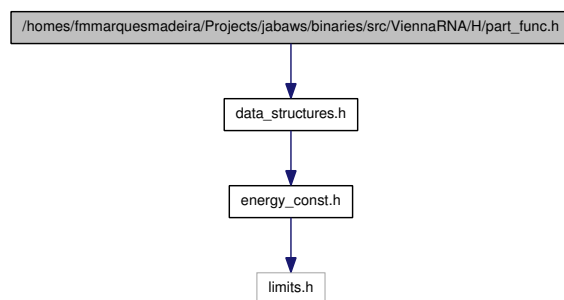
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part\_

func.h File  
Reference

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/p

### func.h File Reference

Partition function of single RNA sequences. Include dependency graph for part\_func.h:



## Functions

- float [pf\\_fold\\_par](#) (const char \*sequence, char \*structure, [pf\\_paramT](#) \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)  
*Compute the partition function  $Q$  for a given RNA sequence.*
- float [pf\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function  $Q$  of an RNA sequence.*
- float [pf\\_circ\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function of a circular RNA sequence.*
- char \* [pbacktrack](#) (char \*sequence)  
*Sample a secondary structure from the Boltzmann ensemble according its probability*  
.
- char \* [pbacktrack\\_circ](#) (char \*sequence)  
*Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.*
- void [free\\_pf\\_arrays](#) (void)  
*Free arrays for the partition function recursions.*
- void [update\\_pf\\_params](#) (int length)  
*Recalculate energy parameters.*
- void [update\\_pf\\_params\\_par](#) (int length, [pf\\_paramT](#) \*parameters)  
*Recalculate energy parameters.*
- double \* [export\\_bppm](#) (void)

*Get a pointer to the base pair probability array*

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

- void `assign_plist_from_pr` (`plist **pl`, `double *probs`, `int length`, `double cutoff`)  
*Create a `plist` from a probability matrix.*
- int `get_pf_arrays` (`short **S_p`, `short **S1_p`, `char **ptype_p`, `double **qb_p`,  
`double **qm_p`, `double **q1k_p`, `double **qln_p`)  
*Get the pointers to (almost) all 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.*
- char \* `get_centroid_struct_pl` (`int length`, `double *dist`, `plist *pl`)  
*Get the centroid structure of the ensemble.*
- char \* `get_centroid_struct_pr` (`int length`, `double *dist`, `double *pr`)  
*Get the centroid structure of the ensemble.*
- double `mean_bp_distance` (`int length`)  
*Get the mean base pair distance of the last partition function computation.*
- double `mean_bp_distance_pr` (`int length`, `double *pr`)  
*Get the mean base pair distance in the thermodynamic ensemble.*
- void `bppm_to_structure` (`char *structure`, `double *pr`, `unsigned int length`)  
*Create a dot-bracket like structure string from base pair probability matrix.*
- char `bppm_symbol` (`const float *x`)  
*Get a pseudo dot bracket notation for a given probability information.*
- void `init_pf_fold` (`int length`)  
*Allocate space for `pf_fold()`.*
- char \* `centroid` (`int length`, `double *dist`)
- double `mean_bp_dist` (`int length`)
- double `expLoopEnergy` (`int u1`, `int u2`, `int type`, `int type2`, `short si1`, `short sj1`,  
`short sp1`, `short sq1`)
- double `expHairpinEnergy` (`int u`, `int type`, `short si1`, `short sj1`, `const char *string`)

## Variables

- int `st_back`  
*Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.*

## 11.23

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part\_  
func.h File

### 11.23.1 Detailed Description

207

Partition function of single RNA sequences. This file includes (almost) all function declarations within the **RNAlib** that are related to Partion function folding...

#### 11.23.2 Function Documentation

##### 11.23.2.1 void init\_pf\_fold (int *length*)

Allocate space for [pf\\_fold\(\)](#).

##### Deprecated

This function is obsolete and will be removed soon!

##### 11.23.2.2 char\* centroid (int *length*, double \* *dist*)

##### Deprecated

This function is deprecated and should not be used anymore as it is not threadsafe!

See also:

[get\\_centroid\\_struct\\_pl\(\)](#), [get\\_centroid\\_struct\\_pr\(\)](#)

##### 11.23.2.3 double mean\_bp\_dist (int *length*)

get the mean pair distance of ensemble

##### Deprecated

This function is not threadsafe and should not be used anymore. Use [mean\\_bp\\_distance\(\)](#) instead!

##### 11.23.2.4 double expLoopEnergy (int *u1*, int *u2*, int *type*, int *type2*, short *si1*, short *sj1*, short *sp1*, short *sq1*)

##### Deprecated

Use [exp\\_E\\_IntLoop\(\)](#) from [loop\\_energies.h](#) instead

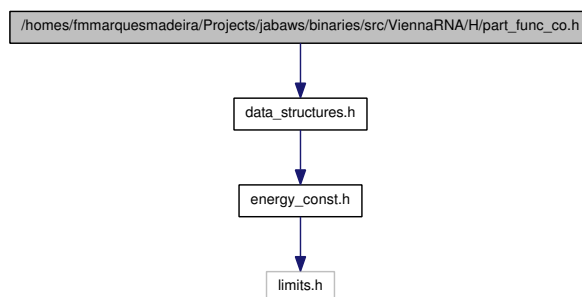
##### 11.23.2.5 double expHairpinEnergy (int *u*, int *type*, short *si1*, short *sj1*, const char \* *string*)

##### Deprecated

Use [exp\\_E\\_Hairpin\(\)](#) from [loop\\_energies.h](#) instead

## 11.24 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/func\_co.h File Reference

Partition function for two RNA sequences. Include dependency graph for part\_func\_co.h:



### Functions

- `cofoldF co_pf_fold` (char \*sequence, char \*structure)  
*Calculate partition function and base pair probabilities.*
- `cofoldF co_pf_fold_par` (char \*sequence, char \*structure, `pf_paramT` \*parameters, int calculate\_bppm, int is\_constrained)  
*Calculate partition function and base pair probabilities.*
- `double * export_co_bppm` (void)  
*Get a pointer to the base pair probability array.*
- `void free_co_pf_arrays` (void)  
*Free the memory occupied by `co_pf_fold()`.*
- `void update_co_pf_params` (int length)  
*Recalculate energy parameters.*
- `void update_co_pf_params_par` (int length, `pf_paramT` \*parameters)  
*Recalculate energy parameters.*
- `void compute_probabilities` (double FAB, double FEA, double FEB, struct `plist` \*prAB, struct `plist` \*prA, struct `plist` \*prB, int Alength)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- `ConcEnt * get_concentrations` (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double \*startconc)  
*Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.*

## 11.24

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part\_  
func\_co.h File

### Reference

209

- `plist * get_plist` (struct `plist *pl`, int `length`, double `cut_off`)
- void `init_co_pf_fold` (int `length`)

### Variables

- int `mirnatog`  
*Toggles no intrabp in 2nd mol.*
- double `F_monomer` [2]  
*Free energies of the two monomers.*

### 11.24.1 Detailed Description

Partition function for two RNA sequences. As for folding one RNA molecule, this computes the partition function of all possible structures and the base pair probabilities. Uses the same global `pf_scale` variable to avoid overflows.

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

After computing the partition functions of all possible dimers 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)

### 11.24.2 Function Documentation

#### 11.24.2.1 `plist* get_plist (struct plist *pl, int length, double cut_off)`

DO NOT USE THIS FUNCTION ANYMORE

#### Deprecated

{ This function is deprecated and will be removed soon!} use `assign_plist_from_pr()` instead!

**11.24.2.2 void init\_co\_pf\_fold (int *length*)**

DO NOT USE THIS FUNCTION ANYMORE

**Deprecated**

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

## 11.25

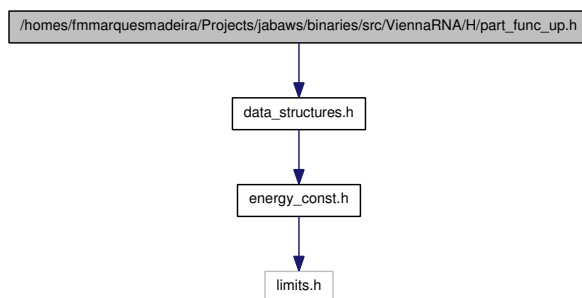
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/part\_

func\_up.h File

11.25 Reference /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/p

### func\_up.h File Reference

Partition Function Cofolding as stepwise process. Include dependency graph for part\_ - func\_up.h:



## Functions

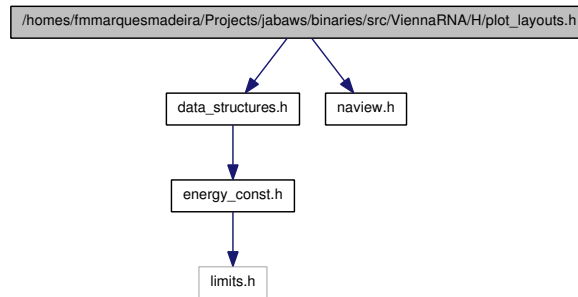
- `pu_contrib * pf_unstru` (`char *sequence`, `int max_w`)  
*Calculate the partition function over all unpaired regions of a maximal length.*
- `interact * pf_interact` (`const char *s1`, `const char *s2`, `pu_contrib *p_c`, `pu_contrib *p_c2`, `int max_w`, `char *cstruc`, `int incr3`, `int incr5`)  
*Calculates the probability of a local interaction between two sequences.*
- `void free_interact` (`interact *pin`)  
*Frees the output of function `pf_interact()`.*
- `void free_pu_contrib_struct` (`pu_contrib *pu`)  
*Frees the output of function `pf_unstru()`.*

### 11.25.1 Detailed Description

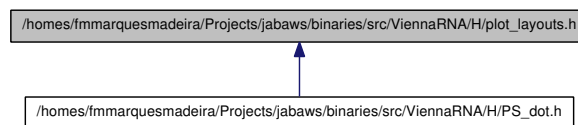
Partition Function Cofolding as stepwise process. In this approach to cofolding the interaction between two RNA molecules is seen as a stepwise process. In a first step, the target molecule has to adopt a structure in which a binding site is accessible. In a second step, the ligand molecule will hybridize with a region accessible to an interaction. Consequently the algorithm is designed as a two step process: The first step is the calculation of the probability that a region within the target is unpaired, or equivalently, the calculation of the free energy needed to expose a region. In the second step we compute the free energy of an interaction for every possible binding site.

## 11.26 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/plot\_layouts.h File Reference

Secondary structure plot layout algorithms. Include dependency graph for plot\_layouts.h:



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



### Defines

- `#define VRNA_PLOT_TYPE_SIMPLE 0`  
*Definition of Plot type simple.*
- `#define VRNA_PLOT_TYPE_NAVIEW 1`  
*Definition of Plot type Naview.*
- `#define VRNA_PLOT_TYPE_CIRCULAR 2`  
*Definition of Plot type Circular.*

### Functions

- `int simple_xy_coordinates` (short \*pair\_table, float \*X, float \*Y)  
*Calculate nucleotide coordinates for secondary structure plot the Simple way.*
- `int simple_circplot_coordinates` (short \*pair\_table, float \*x, float \*y)  
*Calculate nucleotide coordinates for Circular Plot.*



- int `rna_plot_type`

*Switch for changing the secondary structure layout algorithm.*

### 11.26.1 Detailed Description

Secondary structure plot layout algorithms. c Ronny Lorenz The ViennaRNA Package

### 11.26.2 Define Documentation

#### 11.26.2.1 #define VRNA\_PLOT\_TYPE\_SIMPLE 0

Definition of Plot type *simple*. This is the plot type definition for several RNA structure plotting functions telling them to use **Simple** plotting algorithm

**See also:**

[rna\\_plot\\_type](#), [PS\\_rna\\_plot\\_a\(\)](#), [PS\\_rna\\_plot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

#### 11.26.2.2 #define VRNA\_PLOT\_TYPE\_NAVIEW 1

Definition of Plot type *Naview*. This is the plot type definition for several RNA structure plotting functions telling them to use **Naview** plotting algorithm

**See also:**

[rna\\_plot\\_type](#), [PS\\_rna\\_plot\\_a\(\)](#), [PS\\_rna\\_plot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

#### 11.26.2.3 #define VRNA\_PLOT\_TYPE\_CIRCULAR 2

Definition of Plot type *Circular*. This is the plot type definition for several RNA structure plotting functions telling them to produce a **Circular plot**

**See also:**

[rna\\_plot\\_type](#), [PS\\_rna\\_plot\\_a\(\)](#), [PS\\_rna\\_plot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

### 11.26.3 Function Documentation

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

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

**See also:**

[make\\_pair\\_table\(\)](#), [rna\\_plot\\_type](#), [simple\\_circplot\\_coordinates\(\)](#), [naview\\_xy\\_coordinates\(\)](#), [PS\\_rna\\_plot\\_a\(\)](#), [PS\\_rna\\_plot](#), [svg\\_rna\\_plot\(\)](#)

**Parameters:**

*pair\_table* The pair table of the secondary structure

*X* a pointer to an array with enough allocated space to hold the x coordinates

*Y* a pointer to an array with enough allocated space to hold the y coordinates

**Returns:**

length of sequence on success, 0 otherwise

#### 11.26.3.2 `int simple_circplot_coordinates (short * pair_table, float * x, float * y)`

Calculate nucleotide coordinates for *Circular Plot*. This function calculates the coordinates of nucleotides mapped in equal 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.  $P_x^t[i] = X[i] * rs$  and  $P_y^t[i] = Y[i] * rs$ .

**See also:**

[make\\_pair\\_table\(\)](#), [rna\\_plot\\_type](#), [simple\\_xy\\_coordinates\(\)](#), [naview\\_xy\\_coordinates\(\)](#), [PS\\_rna\\_plot\\_a\(\)](#), [PS\\_rna\\_plot](#), [svg\\_rna\\_plot\(\)](#)

**Parameters:**

*pair\_table* The pair table of the secondary structure

*x* a pointer to an array with enough allocated space to hold the x coordinates

*y* a pointer to an array with enough allocated space to hold the y coordinates

**Returns:**

length of sequence on success, 0 otherwise

## 11.26

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/plot\_ -  
layouts.h File

### 11.26.4 Variable Documentation

215

#### 11.26.4.1 int rna\_plot\_type

Switch for changing the secondary structure layout algorithm. Current possibilities are 0 for a simple radial drawing or 1 for the modified radial drawing taken from the *naview* program of Brucoleri & Heinrich (1988).

#### Note:

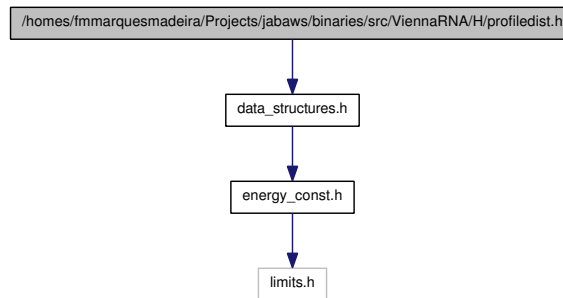
To provide thread safety please do not rely on this global variable in future implementations but pass a plot type flag directly to the function that decides which layout algorithm it may use!

#### See also:

[VRNA\\_PLOT\\_TYPE\\_SIMPLE](#), [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#), [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#)

## 11.27 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/profiledist.h File Reference

Include dependency graph for profiledist.h:



### Functions

- float [profile\\_edit\\_distance](#) (const float \*T1, const float \*T2)  
*Align the 2 probability profiles T1, T2*  
.
- float \* [Make\\_bp\\_profile\\_bppm](#) (double \*bppm, int length)  
*condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.*
- void [print\\_bppm](#) (const float \*T)  
*print string representation of probability profile*
- void [free\\_profile](#) (float \*T)  
*free space allocated in Make\_bp\_profile*
- float \* [Make\\_bp\\_profile](#) (int length)

#### 11.27.1 Detailed Description

#### 11.27.2 Function Documentation

##### 11.27.2.1 float profile\_edit\_distance (const float \* T1, const float \* T2)

Align the 2 probability profiles T1, T2

. This is like a Needleman-Wunsch alignment, we should really use affine gap-costs ala Gotoh

**11.27.2.2** `float* Make_bp_profile_bppm (double * bppm, int length)`

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired. This resulting probability profile is used as input for `profile_edit_distance`

**Parameters:**

*bppm* A pointer to the base pair probability matrix

*length* The length of the sequence

**Returns:**

The bp profile

**11.27.2.3** `void free_profile (float * T)`

free space allocated in `Make_bp_profile` Backward compatibility only. You can just use plain `free()`

**11.27.2.4** `float* Make_bp_profile (int length)`

**Note:**

This function is NOT threadsafe

**See also:**

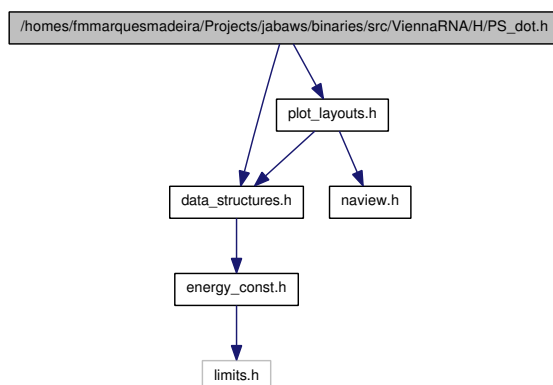
[Make\\_bp\\_profile\\_bppm\(\)](#)

**Deprecated**

This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement

## 11.28 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/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:



### Functions

- int [PS\\_rna\\_plot](#) (char \*string, char \*structure, char \*file)  
*Produce a secondary structure graph in PostScript and write it to 'filename'.*
- int [PS\\_rna\\_plot\\_a](#) (char \*string, char \*structure, char \*file, char \*pre, char \*post)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.*
- int [gmlRNA](#) (char \*string, char \*structure, char \*ssfile, char option)  
*Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.*
- int [ssv\\_rna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure graph in SStructView format.*
- int [svg\\_rna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot in SVG format and write it to a file.*
- int [xrna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot for further editing in XRNA.*
- int [PS\\_dot\\_plot\\_list](#) (char \*seq, char \*filename, [plist](#) \*pl, [plist](#) \*mf, char \*comment)  
*Produce a postscript dot-plot from two pair lists.*

## 11.28

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/PS\_  
dot.h File

### Reference

219

- int [aliPS\\_color\\_aln](#) (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])
- int [PS\\_dot\\_plot](#) (char \*string, char \*file)  
*Produce postscript dot-plot.*

### 11.28.1 Detailed Description

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

### 11.28.2 Function Documentation

#### 11.28.2.1 int PS\_rna\_plot (char \* *string*, char \* *structure*, char \* *file*)

Produce a secondary structure graph in PostScript and write it to 'filename'. Note that this function has changed from previous versions and now expects the structure to be plotted in dot-bracket notation as an argument. It does not make use of the global [base\\_pair](#) array anymore.

#### Parameters:

- string* The RNA sequence
- structure* The secondary structure in dot-bracket notation
- file* The filename of the postscript output

#### Returns:

1 on success, 0 otherwise

#### 11.28.2.2 int PS\_rna\_plot\_a (char \* *string*, char \* *structure*, char \* *file*, char \* *pre*, char \* *post*)

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'. Same as [PS\\_rna\\_plot\(\)](#) but adds extra PostScript macros for various annotations (see generated PS code). The 'pre' and 'post' variables contain PostScript code that is verbatim copied in the resulting PS file just before and after the structure plot. If both arguments ('pre' and 'post') are NULL, no additional macros will be printed into the PostScript.

#### Parameters:

- string* The RNA sequence
- structure* The secondary structure in dot-bracket notation
- file* The filename of the postscript output

*pre* PostScript code to appear before the secondary structure plot

*post* PostScript code to appear after the secondary structure plot

**Returns:**

1 on success, 0 otherwise

**11.28.2.3 int gmlRNA (char \* *string*, char \* *structure*, char \* *ssfile*, char *option*)**

Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file. If 'option' is an uppercase letter the RNA sequence is used to label nodes, if 'option' equals 'X' or 'x' the resulting file will coordinates for an initial layout of the graph.

**Parameters:**

*string* The RNA sequence

*structure* The secondary structure in dot-bracket notation

*ssfile* The filename of the gml output

*option* The option flag

**Returns:**

1 on success, 0 otherwise

**11.28.2.4 int ssv\_rna\_plot (char \* *string*, char \* *structure*, char \* *ssfile*)**

Produce a secondary structure graph in SStructView format. Write coord file for SStructView

**Parameters:**

*string* The RNA sequence

*structure* The secondary structure in dot-bracket notation

*ssfile* The filename of the ssv output

**Returns:**

1 on success, 0 otherwise

**11.28.2.5 int svg\_rna\_plot (char \* *string*, char \* *structure*, char \* *ssfile*)**

Produce a secondary structure plot in SVG format and write it to a file.



## 11.28

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/PS\_

**dot.h File  
Parameters:  
Reference**

221

---

*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

### 11.28.2.6 int xrna\_plot (char \* *string*, char \* *structure*, char \* *ssfile*)

Produce a secondary structure plot for further editing in XRNA.

### Parameters:

*string* The RNA sequence  
*structure* The secondary structure in dot-bracket notation  
*ssfile* The filename of the xrna output

### Returns:

1 on success, 0 otherwise

### 11.28.2.7 int PS\_dot\_plot\_list (char \* *seq*, char \* *filename*, plist \* *pl*, plist \* *mf*, char \* *comment*)

Produce a postscript dot-plot from two pair lists. This function reads two [plist](#) structures (e.g. base pair probabilities and a secondary structure) as produced by [assign\\_plist\\_from\\_pr\(\)](#) and [assign\\_plist\\_from\\_db\(\)](#) and produces a postscript "dot plot" that is written to 'filename'.

Using base pair probabilities in the first and mfe structure in the second [plist](#), the resulting "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy structure.

### See also:

[assign\\_plist\\_from\\_pr\(\)](#), [assign\\_plist\\_from\\_db\(\)](#)

### Parameters:

*seq* The RNA sequence  
*filename* A filename for the postscript output  
*pl* The base pair probability pairlist  
*mf* The mfe secondary structure pairlist  
*comment* A comment

### Returns:

1 if postscript was successfully written, 0 otherwise

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

PS\_color\_aln for duplexes

**11.28.2.9** `int PS_dot_plot (char * string, char * file)`

Produce postscript dot-plot. Wrapper to PS\_dot\_plot\_list

Reads base pair probabilities produced by [pf\\_fold\(\)](#) from the global array [pr](#) and the pair list [base\\_pair](#) produced by [fold\(\)](#) and produces a postscript "dot plot" that is written to 'filename'. The "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy

**Note:**

DO NOT USE THIS FUNCTION ANYMORE SINCE IT IS NOT THREAD-SAFE

**Deprecated**

This function is deprecated and will be removed soon! Use [PS\\_dot\\_plot\\_list\(\)](#) instead!

11.29

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/read\_ -

epars.h File

Reference

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/r

## epars.h File Reference

### Functions

- void [read\\_parameter\\_file](#) (const char fname[ ])  
*Read energy parameters from a file.*
- void [write\\_parameter\\_file](#) (const char fname[ ])  
*Write energy parameters to a file.*

### 11.29.1 Detailed Description

## 11.30 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaR 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.*

**Reference 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.*

**11.30.1 Detailed Description**

Parsing and Coarse Graining of Structures. Example:

```
* .((..(((...)))..((...))). is the bracket or full tree
* becomes expanded: - expand_Full() -
* ((U)((U)(U)(((U)(U)(U)P)P)P)(U)(U)((U)(U)P)P)P)(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)
*
```

**11.30.2 Function Documentation****11.30.2.1 char\* b2HIT (const char \* *structure*)**

Converts the full structure from bracket notation to the HIT notation including root.

**Parameters:**

*structure*

**Returns:**

**11.30.2.2 char\* b2C (const char \* *structure*)**

Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

**Parameters:**

*structure*

**Returns:****11.30.2.3 char\* b2Shapiro (const char \* *structure*)**

Converts the full structure from bracket notation to the *weighted* coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.

**Parameters:**

*structure*

**Returns:****11.30.2.4 char\* add\_root (const char \* *structure*)**

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

**Parameters:**

*structure*

**Returns:****11.30.2.5 char\* expand\_Shapiro (const char \* *coarse*)**

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

**Parameters:**

*coarse*

**Returns:**

Reference

11.30.2.6 char\* expand\_Full (const char \* *structure*)

---

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

**Parameters:**

*structure*

**Returns:**

11.30.2.7 char\* unexpand\_Full (const char \* *ffull*)

Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

**Parameters:**

*ffull*

**Returns:**

11.30.2.8 char\* unweight (const char \* *wcoarse*)

Strip weights from any weighted tree.

**Parameters:**

*wcoarse*

**Returns:**

11.30.2.9 void unexpand\_aligned\_F (char \* *align*[2])

Converts two aligned structures in expanded notation. Takes two aligned structures as produced by [tree\\_edit\\_distance\(\)](#) function back to bracket notation with '\_' as the gap character. The result overwrites the input.

**Parameters:**

*align*

**11.30.2.10 void parse\_structure (const char \* *structure*)**

Collects a statistic of structure elements of the full structure in bracket notation. The function writes to the following global variables: [loop\\_size](#), [loop\\_degree](#), [helix\\_size](#), [loops](#), [pairs](#), [unpaired](#)

**Parameters:**

*structure*

**Returns:**

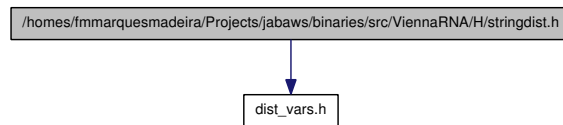


11.31 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/stringdist.h File

Reference

## 11.31 <sup>229</sup> /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/stringdist.h File Reference

Functions for String Alignment. Include dependency graph for stringdist.h:



### Functions

- `swString * Make_swString (char *string)`  
*Convert a structure into a format suitable for `string_edit_distance()`.*
- `float string_edit_distance (swString *T1, swString *T2)`  
*Calculate the string edit distance of T1 and T2.*

#### 11.31.1 Detailed Description

Functions for String Alignment.

#### 11.31.2 Function Documentation

##### 11.31.2.1 `swString* Make_swString (char * string)`

Convert a structure into a format suitable for `string_edit_distance()`.

##### Parameters:

*string*

##### Returns:

##### 11.31.2.2 `float string_edit_distance (swString * T1, swString * T2)`

Calculate the string edit distance of T1 and T2.

##### Parameters:

*T1*

*T2*

**Returns:**

## 11.32

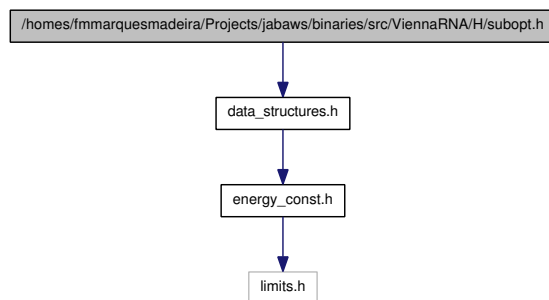
/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/subopt.h

File Reference

11.32 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/s<sup>231</sup>

### File Reference

RNAsubopt and density of states declarations. Include dependency graph for subopt.h:



### Functions

- **SOLUTION** \* `subopt` (char \*seq, char \*structure, int delta, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- **SOLUTION** \* `subopt_par` (char \*seq, char \*structure, **paramT** \*parameters, int delta, int is\_constrained, int is\_circular, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- **SOLUTION** \* `subopt_circ` (char \*seq, char \*sequence, int delta, FILE \*fp)  
*Returns list of circular subopt structures or writes to fp.*

### Variables

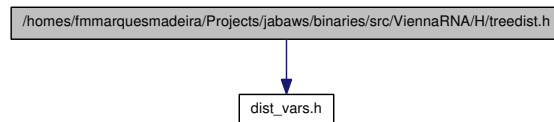
- int `subopt_sorted`  
*Sort output by energy.*
- double `print_energy`  
*printing threshold for use with logML*
- int `density_of_states` [MAXDOS+1]  
*The Density of States.*

#### 11.32.1 Detailed Description

RNAsubopt and density of states declarations.

## 11.33 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA 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.*

#### 11.33.1 Detailed Description

Functions for [Tree](#) Edit Distances.

#### 11.33.2 Function Documentation

##### 11.33.2.1 [Tree](#)\* `make_tree` (`char * struc`)

Constructs a [Tree](#) ( essentially the postorder list ) of the structure 'struc', for use in `tree_edit_distance()`.

##### Parameters:

*struc* may be any rooted structure representation.

##### Returns:

---

11.33.2.2 float tree\_edit\_distance (Tree \* *T1*, Tree \* *T2*)

Calculates the edit distance of the two trees.

**Parameters:**

*T1*

*T2*

**Returns:**

11.33.2.3 void free\_tree (Tree \* *t*)

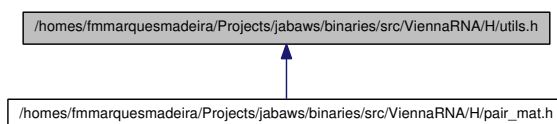
Free the memory allocated for Tree *t*.

**Parameters:**

*t*

## 11.34 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Utils.h File Reference

Various utility- and helper-functions used throughout the Vienna RNA package. This graph shows which files directly or indirectly include this file:



### Defines

- #define `VRNA_INPUT_ERROR` 1U
- #define `VRNA_INPUT_QUIT` 2U
- #define `VRNA_INPUT_MISC` 4U
- #define `VRNA_INPUT_FASTA_HEADER` 8U
- #define `VRNA_INPUT_SEQUENCE` 16U
- #define `VRNA_INPUT_CONSTRAINT` 32U
- #define `VRNA_INPUT_NO_TRUNCATION` 256U
- #define `VRNA_INPUT_NO_REST` 512U
- #define `VRNA_INPUT_NO_SPAN` 1024U
- #define `VRNA_INPUT_NOSKIP_BLANK_LINES` 2048U
- #define `VRNA_INPUT_BLANK_LINE` 4096U
- #define `VRNA_INPUT_NOSKIP_COMMENTS` 128U
- #define `VRNA_INPUT_COMMENT` 8192U
- #define `VRNA_CONSTRAINT_PIPE` 1U
- #define `VRNA_CONSTRAINT_DOT` 2U
- #define `VRNA_CONSTRAINT_X` 4U
- #define `VRNA_CONSTRAINT_ANG_BRACK` 8U
- #define `VRNA_CONSTRAINT_RND_BRACK` 16U
- #define `VRNA_CONSTRAINT_MULTILINE` 32U
- #define `VRNA_CONSTRAINT_NO_HEADER` 64U
- #define `VRNA_CONSTRAINT_ALL` 128U
- #define `VRNA_CONSTRAINT_G` 256U
- #define `VRNA_OPTION_MULTILINE` 32U
- #define `MIN2(A, B)` ((A) < (B) ? (A) : (B))
- #define `MAX2(A, B)` ((A) > (B) ? (A) : (B))
- #define `MIN3(A, B, C)` (MIN2( (MIN2((A),(B))) ,(C)))
- #define `MAX3(A, B, C)` (MAX2( (MAX2((A),(B))) ,(C)))
- #define `XSTR(s)` STR(s)
- #define `STR(s)` #s
- #define `FILENAME_MAX_LENGTH` 80

*Maximum length of filenames that are generated by our programs.*

- 
- #define `FILENAME_ID_LENGTH` 42

*Maximum length of id taken from fasta header for filename generation.*

## Functions

- void \* `space` (unsigned size)  
*Allocate space safely.*
- void \* `xrealloc` (void \*p, unsigned size)  
*Reallocate space safely.*
- void `nrerror` (const char message[ ])  
*Die with an error message.*
- void `warn_user` (const char message[ ])  
*Print a warning message.*
- void `init_rand` (void)  
*Make random number seeds.*
- double `urn` (void)  
*get a random number from [0..1]*
- int `int_urn` (int from, int to)  
*Generates a pseudo random integer in a specified range.*
- char \* `time_stamp` (void)  
*Get a timestamp.*
- char \* `random_string` (int l, const char symbols[ ])  
*Create a random string using characters from a specified symbol set.*
- int `hamming` (const char \*s1, const char \*s2)  
*Calculate hamming distance between two sequences.*
- int `hamming_bound` (const char \*s1, const char \*s2, int n)  
*Calculate hamming distance between two sequences up to a specified length.*
- char \* `get_line` (FILE \*fp)  
*Read a line of arbitrary length from a stream.*
- unsigned int `get_input_line` (char \*\*string, unsigned int options)
- unsigned int `read_record` (char \*\*header, char \*\*sequence, char \*\*\*rest, unsigned int options)  
*Get a data record from stdin.*

- char \* [pack\\_structure](#) (const char \*struc)  
*Pack secondary secondary structure, 5:1 compression using base 3 encoding.*
- char \* [unpack\\_structure](#) (const char \*packed)  
*Unpack secondary structure previously packed with [pack\\_structure\(\)](#).*
- short \* [make\\_pair\\_table](#) (const char \*structure)  
*Create a pair table of a secondary structure.*
- short \* [copy\\_pair\\_table](#) (const short \*pt)  
*Get an exact copy of a pair table.*
- short \* [alimake\\_pair\\_table](#) (const char \*structure)
- short \* [make\\_pair\\_table\\_snoop](#) (const char \*structure)
- int \* [make\\_loop\\_index\\_pt](#) (short \*pt)  
*Compute the "base pair" distance between two secondary structures s1 and s2.*
- void [print\\_tty\\_input\\_seq](#) (void)  
*Print a line to stdout that asks for an input sequence.*
- void [print\\_tty\\_input\\_seq\\_str](#) (const char \*s)  
*Print a line with a user defined string and a ruler to stdout.*
- void [print\\_tty\\_constraint\\_full](#) (void)  
*Print structure constraint characters to stdout (full constraint support).*
- void [print\\_tty\\_constraint](#) (unsigned int option)  
*Print structure constraint characters to stdout. (constraint support is specified by option parameter).*
- void [str\\_DNA2RNA](#) (char \*sequence)  
*Convert a DNA input sequence to RNA alphabet.*
- void [str\\_uppercase](#) (char \*sequence)  
*Convert an input sequence to uppercase.*
- int \* [get\\_iindx](#) (unsigned int length)  
*Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.*
- int \* [get\\_indx](#) (unsigned int length)  
*Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.*
- void [constrain\\_ptypes](#) (const char \*constraint, unsigned int length, char \*ptype, int \*BP, int min\_loop\_size, unsigned int idx\_type)  
*Insert constraining pair types according to constraint structure string.*



- unsigned short `xsubi` [3]

*Current 48 bit random number.*

### 11.34.1 Detailed Description

Various utility- and helper-functions used throughout the Vienna RNA package.

### 11.34.2 Define Documentation

#### 11.34.2.1 `#define VRNA_INPUT_ERROR 1U`

Output flag of `get_input_line()`: "An ERROR has occurred, maybe EOF"

#### 11.34.2.2 `#define VRNA_INPUT_QUIT 2U`

Output flag of `get_input_line()`: "the user requested quitting the program"

#### 11.34.2.3 `#define VRNA_INPUT_MISC 4U`

Output flag of `get_input_line()`: "something was read"

#### 11.34.2.4 `#define VRNA_INPUT_FASTA_HEADER 8U`

Input/Output flag of `get_input_line()`:

if used as input option this tells `get_input_line()` that the data to be read should comply with the FASTA format

the function will return this flag if a fasta header was read

#### 11.34.2.5 `#define VRNA_INPUT_SEQUENCE 16U`

Input flag for `get_input_line()`:

Tell `get_input_line()` that we assume to read a nucleotide sequence

#### 11.34.2.6 `#define VRNA_INPUT_CONSTRAINT 32U`

Input flag for `get_input_line()`:

Tell `get_input_line()` that we assume to read a structure constraint

**11.34.2.7 #define VRNA\_INPUT\_NO\_TRUNCATION 256U**

Input switch for [get\\_input\\_line\(\)](#): "do not truncate the line by eliminating white spaces at end of line"

**11.34.2.8 #define VRNA\_INPUT\_NO\_REST 512U**

Input switch for [read\\_record\(\)](#): "do fill rest array"

**11.34.2.9 #define VRNA\_INPUT\_NO\_SPAN 1024U**

Input switch for [read\\_record\(\)](#): "never allow data to span more than one line"

**11.34.2.10 #define VRNA\_INPUT\_NOSKIP\_BLANK\_LINES 2048U**

Input switch for [read\\_record\(\)](#): "do not skip empty lines"

**11.34.2.11 #define VRNA\_INPUT\_BLANK\_LINE 4096U**

Output flag for [read\\_record\(\)](#): "read an empty line"

**11.34.2.12 #define VRNA\_INPUT\_NOSKIP\_COMMENTS 128U**

Input switch for [get\\_input\\_line\(\)](#): "do not skip comment lines"

**11.34.2.13 #define VRNA\_INPUT\_COMMENT 8192U**

Output flag for [read\\_record\(\)](#): "read a comment"

**11.34.2.14 #define VRNA\_CONSTRAINT\_PIPE 1U**

pipe sign '|' switch for structure constraints (paired with another base)

**11.34.2.15 #define VRNA\_CONSTRAINT\_DOT 2U**

dot '.' switch for structure constraints (no constraint at all)

**11.34.2.16 #define VRNA\_CONSTRAINT\_X 4U**

'x' switch for structure constraint (base must not pair)

## 11.34

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utils.h

File Reference

239

**11.34.2.17 #define VRNA\_CONSTRAINT\_ANG\_BRACK 8U**

angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)

**11.34.2.18 #define VRNA\_CONSTRAINT\_RND\_BRACK 16U**

round brackets '(', ')' switch for structure constraint (base i pairs base j)

**11.34.2.19 #define VRNA\_CONSTRAINT\_MULTILINE 32U**

constraint may span over several lines

**11.34.2.20 #define VRNA\_CONSTRAINT\_NO\_HEADER 64U**

do not print the header information line

**11.34.2.21 #define VRNA\_CONSTRAINT\_ALL 128U**

placeholder for all constraining characters

**11.34.2.22 #define VRNA\_CONSTRAINT\_G 256U**

'+' switch for structure constraint (base is involved in a gquad)

**11.34.2.23 #define VRNA\_OPTION\_MULTILINE 32U**

Tell a function that an input is assumed to span several lines if used as input-option  
A function might also be returning this state telling that it has read data from multiple lines.

**See also:**

`extract_record_rest_structure()`, `read_record()`, `getConstraint()`

**11.34.2.24 #define MIN2(A, B) ((A) < (B) ? (A) : (B))**

Get the minimum of two comparable values

**11.34.2.25 #define MAX2(A, B) ((A) > (B) ? (A) : (B))**

Get the maximum of two comparable values

**11.34.2.26 #define MIN3(A, B, C) (MIN2( (MIN2((A),(B))) ,(C)))**

Get the minimum of three comparable values

**11.34.2.27 #define MAX3(A, B, C) (MAX2( (MAX2((A),(B))) ,(C)))**

Get the maximum of three comparable values

**11.34.2.28 #define XSTR(s) STR(s)**

Stringify a macro after expansion

**11.34.2.29 #define STR(s) #s**

Stringify a macro argument

**11.34.2.30 #define FILENAME\_MAX\_LENGTH 80**

Maximum length of filenames that are generated by our programs. This definition should be used throughout the complete ViennaRNA package wherever a static array holding filenames of output files is declared.

**11.34.2.31 #define FILENAME\_ID\_LENGTH 42**

Maximum length of id taken from fasta header for filename generation. this has to be smaller than FILENAME\_MAX\_LENGTH since in most cases, some suffix will be appended to the ID

**11.34.3 Function Documentation****11.34.3.1 void\* space (unsigned size)**

Allocate space safely.

**Parameters:**

*size* The size of the memory to be allocated in bytes

**Returns:**

A pointer to the allocated memory

## 11.34

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utils.h

### File Reference

241

---

#### 11.34.3.2 void\* xrealloc (void \**p*, unsigned *size*)

Reallocate space safely.

#### Parameters:

*p* A pointer to the memory region to be reallocated

*size* The size of the memory to be allocated in bytes

#### Returns:

A pointer to the newly allocated memory

#### 11.34.3.3 void nerror (const char *message*[ ])

Die with an error message.

#### See also:

[warn\\_user\(\)](#)

#### Parameters:

*message* The error message to be printed before exiting with 'FAILURE'

#### 11.34.3.4 void warn\_user (const char *message*[ ])

Print a warning message. Print a warning message to *stderr*

#### Parameters:

*message* The warning message

#### 11.34.3.5 double urn (void)

get a random number from [0..1]

#### Note:

Usually implemented by calling *erand48()*.

#### Returns:

A random number in range [0..1]

**11.34.3.6 int int\_urn (int from, int to)**

Generates a pseudo random integer in a specified range.

**Parameters:**

*from* The first number in range

*to* The last number in range

**Returns:**

A pseudo random number in range [from, to]

**11.34.3.7 char\* time\_stamp (void)**

Get a timestamp. Returns a string containing the current date in the format

```
Fri Mar 19 21:10:57 1993
```

**Returns:**

A string containing the timestamp

**11.34.3.8 char\* random\_string (int l, const char symbols[ ])**

Create a random string using characters from a specified symbol set.

**Parameters:**

*l* The length of the sequence

*symbols* The symbol set

**Returns:**

A random string of length 'l' containing characters from the symbolset

**11.34.3.9 int hamming (const char \* s1, const char \* s2)**

Calculate hamming distance between two sequences. Calculate the number of positions in which

**Parameters:**

*s1* The first sequence

*s2* The second sequence

**Returns:**

The hamming distance between s1 and s2

## 11.34

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utils.h

File Reference

243

### 11.34.3.10 `int hamming_bound (const char * s1, const char * s2, int n)`

Calculate hamming distance between two sequences up to a specified length. This function is similar to `hamming()` but instead of comparing both sequences up to their actual length only the first '*n*' characters are taken into account

#### Parameters:

*s1* The first sequence

*s2* The second sequence

#### Returns:

The hamming distance between *s1* and *s2*

### 11.34.3.11 `char* get_line (FILE * fp)`

Read a line of arbitrary length from a stream. Returns a pointer to the resulting string. The necessary memory is allocated and should be released using `free()` when the string is no longer needed.

#### Parameters:

*fp* A file pointer to the stream where the function should read from

#### Returns:

A pointer to the resulting string

### 11.34.3.12 `unsigned int get_input_line (char ** string, unsigned int options)`

Retrieve a line from 'stdin' savely while skipping comment characters and other features This function returns the type of input it has read if recognized. An option argument allows to switch between different reading modes.

Currently available options are:

`VRNA_INPUT_NOPRINT_COMMENTS`, `VRNA_INPUT_NOSKIP_COMMENTS`,  
`VRNA_INPUT_NOELIM_WS_SUFFIX`

pass a collection of options as one value like this:

```
get_input_line(string, option_1 | option_2 | option_n)
```

If the function recognizes the type of input, it will report it in the return value. It also reports if a user defined 'quit' command (@-sign on 'stdin') was given. Possible return values are:

`VRNA_INPUT_FASTA_HEADER`, `VRNA_INPUT_ERROR`, `VRNA_INPUT_-MISC`, `VRNA_INPUT_QUIT`

**Parameters:**

*string* A pointer to the character array that contains the line read  
*options* A collection of options for switching the functions behavior

**Returns:**

A flag with information about what has been read

**11.34.3.13 unsigned int read\_record (char \*\* header, char \*\* sequence, char \*\*\* rest, unsigned int options)**

Get a data record from stdin. This function may be used to obtain complete datasets from stdin. A dataset is always defined to contain at least a sequence. If data on stdin starts with a fasta header, i.e. a line like

```
>some header info
```

then `read_record()` will assume that the sequence that follows the header may span over several lines. To disable this behavior and to assign a single line to the argument 'sequence' one can pass `VRNA_INPUT_NO_SPAN` in the 'options' argument. If no fasta header is read in the beginning of a data block, a sequence must not span over multiple lines!

Unless the options `VRNA_INPUT_NOSKIP_COMMENTS` or `VRNA_INPUT_NOSKIP_BLANK_LINES` are passed, a sequence may be interrupted by lines starting with a comment character or empty lines.

A sequence is regarded as completely read if it was either assumed to not span over multiple lines, a secondary structure or structure constraint follows the sequence on the next line or a new header marks the beginning of a new sequence...

All lines following the sequence (this includes comments) and not initiating a new dataset are available through the line-array 'rest'. Here one can usually find the structure constraint or other information belonging to the current dataset. Filling of 'rest' may be prevented by passing `VRNA_INPUT_NO_REST` to the options argument.

**Note:**

This function will exit any program with an error message if no sequence could be read!

The main purpose of this function is to be able to easily parse blocks of data from stdin in the header of a loop where all calculations for the appropriate data is done inside the loop. The loop may be then left on certain return values, e.g.:

```
char *id, *seq, **rest;
int i;
while (!(read_record(&id, &seq, &rest, 0) & (VRNA_INPUT_ERROR | VRNA_INPUT_QUIT))) {
    if(id) printf("%s\n", id);
    printf("%s\n", seq);
    if(rest)
        for(i=0;rest[i];i++)
            printf("%s\n", rest[i]);
}
```



### 11.34

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utils.h

#### File Reference

245

In the example above, the while loop will be terminated when `read_record()` returns either an error or a user initiated quit request.

As long as data is read from stdin, the id is printed if it is available for the current block of data. The sequence will be printed in any case and if some more lines belong to the current block of data each line will be printed as well.

#### Note:

Do not forget to free the memory occupied by header, sequence and rest!

#### Parameters:

- header* A pointer which will be set such that it points to the header of the record
- sequence* A pointer which will be set such that it points to the sequence of the record
- rest* A pointer which will be set such that it points to an array of lines which also belong to the record
- options* Some options which may be passed to alter the behavior of the function, use 0 for no options

#### Returns:

A flag with information about what the function actually did read

#### 11.34.3.14 `char* pack_structure (const char * struc)`

Pack secondary structure, 5:1 compression using base 3 encoding. Returns a binary string encoding of the secondary structure using a 5:1 compression scheme. The string is NULL terminated and can therefore be used with standard string functions such as `strcmp()`. Useful for programs that need to keep many structures in memory.

#### Parameters:

*struc* The secondary structure in dot-bracket notation

#### Returns:

The binary encoded structure

#### 11.34.3.15 `char* unpack_structure (const char * packed)`

Unpack secondary structure previously packed with `pack_structure()`. Translate a compressed binary string produced by `pack_structure()` back into the familiar dot-bracket notation.

#### Parameters:

*packed* The binary encoded packed secondary structure

**Returns:**

The unpacked secondary structure in dot-bracket notation

**11.34.3.16 short\* make\_pair\_table (const char \* structure)**

Create a pair table of a secondary structure. Returns a newly allocated table, such that table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

**Parameters:**

*structure* The secondary structure in dot-bracket notation

**Returns:**

A pointer to the created pair\_table

**11.34.3.17 short\* copy\_pair\_table (const short \* pt)**

Get an exact copy of a pair table.

**Parameters:**

*pt* The pair table to be copied

**Returns:**

A pointer to the copy of 'pt'

**11.34.3.18 short\* alimake\_pair\_table (const char \* structure)**

Pair table for snoop align

**11.34.3.19 short\* make\_pair\_table\_snoop (const char \* structure)**

returns a newly allocated table, such that: table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure. The special pseudoknotted H/ACA-mRNA structure is taken into account.

**11.34.3.20 int\* make\_loop\_index\_pt (short \* pt)**

Compute the "base pair" distance between two secondary structures s1 and s2. The sequences should have the same length. dist = number of base pairs in one structure but not in the other same as edit distance with open-pair close-pair as move-set

**Parameters:**

*str1* First structure in dot-bracket notation

## 11.34

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utils.h

### File Reference

247

---

*str2* Second structure in dot-bracket notation

#### Returns:

The base pair distance between *str1* and *str2*

#### 11.34.3.21 void print\_tty\_input\_seq (void)

Print a line to *stdout* that asks for an input sequence. There will also be a ruler (scale line) printed that helps orientation of the sequence positions

#### 11.34.3.22 void print\_tty\_input\_seq\_str (const char \* s)

Print a line with a user defined string and a ruler to *stdout*. (usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

#### Parameters:

*s* A user defined string that will be printed to *stdout*

#### 11.34.3.23 void print\_tty\_constraint (unsigned int option)

Print structure constraint characters to *stdout*. (constraint support is specified by option parameter). Currently available options are:

[VRNA\\_CONSTRAINT\\_PIPE](#) (paired with another base)

[VRNA\\_CONSTRAINT\\_DOT](#) (no constraint at all)

[VRNA\\_CONSTRAINT\\_X](#) (base must not pair)

[VRNA\\_CONSTRAINT\\_ANG\\_BRACK](#) (paired downstream/upstream)

[VRNA\\_CONSTRAINT\\_RND\\_BRACK](#) (base *i* pairs base *j*)

pass a collection of options as one value like this:

```
print_tty_constraint(option_1 | option_2 | option_n)
```

#### Parameters:

*option* Option switch that tells which constraint help will be printed

#### 11.34.3.24 void str\_DNA2RNA (char \* sequence)

Convert a DNA input sequence to RNA alphabet. This function substitutes *T* and *t* with *U* and *u*, respectively

#### Parameters:

*sequence* The sequence to be converted

**11.34.3.25 void str\_uppercase (char \* *sequence*)**

Convert an input sequence to uppercase.

**Parameters:**

*sequence* The sequence to be converted

**11.34.3.26 int\* get\_iindx (unsigned int *length*)**

Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions. Access of a position "(i,j)" is then accomplished by using

$(i, j) \sim \text{iindx}[i]-j$

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNAPackage

Consult the implemented code to find out about the mapping formula ;)

**See also:**

[get\\_indx\(\)](#)

**Parameters:**

*length* The length of the RNA sequence

**Returns:**

The mapper array

**11.34.3.27 int\* get\_indx (unsigned int *length*)**

Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions. Access of a position "(i,j)" is then accomplished by using

$(i, j) \sim \text{indx}[j]+i$

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNAPackage

Consult the implemented code to find out about the mapping formula ;)

**See also:**

[get\\_iindx\(\)](#)

**Parameters:**

*length* The length of the RNA sequence

**Returns:**

The mapper array

## 11.34

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/utils.h

File Reference

249

**11.34.3.28** `void constrain_ptypes (const char * constraint, unsigned int length,  
char * ptype, int * BP, int min_loop_size, unsigned int idx_type)`

Insert constraining pair types according to constraint structure string.

See also:

[get\\_indx\(\)](#), [get\\_iindx\(\)](#)

Parameters:

*constraint* The structure constraint string

*length* The actual length of the sequence (constraint may be shorter)

*ptype* A pointer to the basepair type array

*min\_loop\_size* The minimal loop size (usually **TURN** )

*idx\_type* Define the access type for base pair type array (0 = indx, 1 = iindx)

### 11.34.4 Variable Documentation

#### 11.34.4.1 unsigned short xsubi[3]

Current 48 bit random number. This variable is used by [urn\(\)](#). These should be set to some random number seeds before the first call to [urn\(\)](#).

See also:

[urn\(\)](#)

## 11.35 /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/epars.h File Reference

Free energy parameters for parameter file conversion.

### 11.35.1 Detailed Description

Free energy parameters for parameter file conversion. This file contains the free energy parameters used in ViennaRNAPackage 1.8.4. They are summarized in:

D.H.Mathews, J. Sabina, M. ZUker, D.H. Turner "Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure" *JMB*, 288, pp 911-940, 1999

Enthalpies taken from:

A. Walter, D Turner, J Kim, M Lyttle, P M"uller, D Mathews, M Zuker "Coaxial stacking 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-Neighbor 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

11.36

/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/lib/1.8.4\_ -

intloops.h File

Reference /homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/lib/

## **intloops.h File Reference**

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

### **11.36.1 Detailed Description**

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

# Index

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Project/](#)  
153 [energies.h](#), 191

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Project/](#)  
155 [fold.h](#), 201

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Project/](#)  
198 [fold.h](#), 202

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Project/](#)  
190 [fold.h](#), 203

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/Project/](#)  
200 [func.h](#), 205

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/PS\\_](#)  
[dot.h](#), 218 [func\\_co.h](#), 208

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/RNAstruct.h](#),  
224 [func\\_up.h](#), 211

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/allfold.h](#),  
156 [layouts.h](#), 212

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/cofold.h](#),  
159 [216](#)

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/convert\\_](#)  
[epars.h](#), 161 [epars.h](#), 223

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/data\\_](#)  
[structures.h](#), 162 [229](#)

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/dist\\_](#)  
[vars.h](#), 165 [231](#)

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/duplex.h](#),  
167 [232](#)

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/edit\\_](#)  
[cost.h](#), 168 [234](#)

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/energy\\_](#)  
[const.h](#), 169 [epars.h](#), 250

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/findpath.h](#),  
171 [initops.h](#), 251

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold.h](#),  
173 [RNAstruct.h](#), 226

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/fold\\_](#)  
[vars.h](#), 180 [consensus\\_mfe\\_fold](#), 71

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/gquad.h](#),  
186 [update\\_alifold\\_params](#), 158

[/homes/fmmarquesmadeira/Projects/jabaws/binaries/src/ViennaRNA/H/inverse.h](#),  
189 [local\\_consensus\\_fold](#), 81



- alimake\_pair\_table
  - utils.h, 246
- alipbacktrack
  - consensus\_stochbt, 75
- alipf\_circ\_fold
  - consensus\_pf\_fold, 74
- alipf\_fold
  - consensus\_pf\_fold, 73
- alipf\_fold\_par
  - consensus\_pf\_fold, 73
- aliPS\_color\_aln
  - PS\_dot.h, 221
- alloc\_sequence\_arrays
  - consensus\_fold, 68
- alpha
  - pf\_paramT, 134
- assign\_plist\_from\_db
  - fold.h, 177
- assign\_plist\_from\_pr
  - pf\_fold, 42
- b2C
  - RNAstruct.h, 225
- b2HIT
  - RNAstruct.h, 225
- b2Shapiro
  - RNAstruct.h, 226
- backtrack\_GQuad\_IntLoop
  - gquad.h, 187
- backtrack\_GQuad\_IntLoop\_L
  - gquad.h, 187
- backtrack\_type
  - fold\_vars.h, 185
- base\_pair
  - fold\_vars.h, 184
- bondT, 111
- bondTEn, 112
- BONUS
  - energy\_const.h, 170
- Calculate Partition Functions of a Distance Based Partitioning, 104
- Calculate Secondary Structures of two RNAs upon Dimerization, 54
- Calculating MFE representatives of a Distance Based Partitioning, 101
- Calculating Minimum Free Energy (MFE) Structures, 33
- Calculating Partition Functions and Pair Probabilities, 37
- centroid
  - part\_func.h, 207
- centroid\_fold
  - get\_centroid\_struct\_pl, 46
  - get\_centroid\_struct\_pr, 46
- Change and Precalculate Energy Parameter Sets and Boltzmann Factors, 82
- circalifold
  - consensus\_mfe\_fold, 71
- circfold
  - mfe\_fold, 35
- Classified Dynamic Programming, 99
- co\_pf\_fold
  - pf\_cofold, 59
- co\_pf\_fold\_par
  - pf\_cofold, 59
- cofold
  - mfe\_cofold, 55
- cofold.h
  - get\_monomere\_mfes, 160
  - initialize\_cofold, 160
- cofoldF, 113
- Compute the centroid structure, 46
- Compute the Density of States, 109
- Compute the structure with maximum expected accuracy (MEA), 45
- compute\_probabilities
  - pf\_cofold, 61
- ConcEnt, 114
- consensus\_fold
  - alloc\_sequence\_arrays, 68
  - cv\_fact, 70
  - encode\_ali\_sequence, 68
  - energy\_of\_alistruct, 67
  - free\_sequence\_arrays, 69
  - get\_alipf\_arrays, 69
  - get\_mpi, 67
  - nc\_fact, 70
- consensus\_mfe\_fold
  - alifold, 71
  - circalifold, 71
- consensus\_pf\_fold
  - alipf\_circ\_fold, 74
  - alipf\_fold, 73
  - alipf\_fold\_par, 73
  - export\_ali\_bppm, 74
- consensus\_stochbt
  - alipbacktrack, 75
- constrain, 115

- constrain\_ptypes
  - utils.h, 248
- convert\_parameter\_file
  - energy\_parameters\_convert, 91
- Converting energy parameter files, 88
- COORDINATE, 116
- copy\_pair\_table
  - utils.h, 246
- cost\_matrix
  - dist\_vars.h, 165
- cpair, 117
- cut\_point
  - fold\_vars.h, 183
- cv\_fact
  - consensus\_fold, 70
- dangles
  - fold\_vars.h, 182
  - model\_detailsT, 126
- density\_of\_states
  - dos, 109
- destroy\_TwoDfold\_variables
  - kl\_neighborhood\_mfe, 102
- destroy\_TwoDpfold\_variables
  - kl\_neighborhood\_pf, 105
- dist\_vars.h
  - cost\_matrix, 165
  - edit\_backtrack, 165
- Distance based partitioning of the Secondary Structure Space, 100
- do\_backtrack
  - fold\_vars.h, 184
- dos
  - density\_of\_states, 109
- duplexT, 118
- dupVar, 119
- E\_Hairpin
  - loop\_energies.h, 193
- E\_IntLoop
  - loop\_energies.h, 192
- E\_Stem
  - loop\_energies.h, 194
- edit\_backtrack
  - dist\_vars.h, 165
- encode\_ali\_sequence
  - consensus\_fold, 68
- Energy evaluation, 93
- energy\_const.h
  - BONUS, 170
  - FORBIDDEN, 169
  - GASCONST, 169
  - INF, 169
  - K0, 169
  - MAXLOOP, 170
  - NBPAIRS, 170
  - TURN, 170
- energy\_of\_alistruct
  - consensus\_fold, 67
- energy\_of\_circ\_struct
  - fold.h, 179
- energy\_of\_circ\_struct\_par
  - eval, 95
- energy\_of\_circ\_structure
  - eval, 94
- energy\_of\_move
  - fold.h, 175
- energy\_of\_move\_pt
  - fold.h, 176
- energy\_of\_struct
  - fold.h, 177
- energy\_of\_struct\_par
  - eval, 94
- energy\_of\_struct\_pt
  - fold.h, 178
- energy\_of\_struct\_pt\_par
  - eval, 96
- energy\_of\_structure
  - eval, 94
- energy\_of\_structure\_pt
  - eval, 95
- energy\_parameters
  - get\_boltzmann\_factor\_copy, 84
  - get\_boltzmann\_factors, 84
  - get\_scaled\_parameters, 83
  - get\_scaled\_pf\_parameters, 83
  - scale\_parameters, 83
- energy\_parameters\_convert
  - convert\_parameter\_file, 91
  - VRNA\_CONVERT\_OUTPUT\_-ALL, 89
  - VRNA\_CONVERT\_OUTPUT\_-BULGE, 90
  - VRNA\_CONVERT\_OUTPUT\_-DANGLE3, 90
  - VRNA\_CONVERT\_OUTPUT\_-DANGLE5, 89
  - VRNA\_CONVERT\_OUTPUT\_-DUMP, 91

- VRNA\_CONVERT\_OUTPUT\_HP, 89
- VRNA\_CONVERT\_OUTPUT\_INT, 90
- VRNA\_CONVERT\_OUTPUT\_INT\_11, 90
- VRNA\_CONVERT\_OUTPUT\_INT\_21, 90
- VRNA\_CONVERT\_OUTPUT\_INT\_22, 90
- VRNA\_CONVERT\_OUTPUT\_MISC, 90
- VRNA\_CONVERT\_OUTPUT\_ML, 90
- VRNA\_CONVERT\_OUTPUT\_MM\_EXT, 89
- VRNA\_CONVERT\_OUTPUT\_MM\_HP, 89
- VRNA\_CONVERT\_OUTPUT\_MM\_INT, 89
- VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N, 89
- VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23, 89
- VRNA\_CONVERT\_OUTPUT\_MM\_MULTI, 89
- VRNA\_CONVERT\_OUTPUT\_NINIO, 91
- VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP, 90
- VRNA\_CONVERT\_OUTPUT\_STACK, 89
- VRNA\_CONVERT\_OUTPUT\_VANILLA, 90
- energy\_parameters\_rw
  - read\_parameter\_file, 86
  - write\_parameter\_file, 86
- energy\_set
  - fold\_vars.h, 183
- Enumerating Suboptimal Structures, 48
- eval
  - energy\_of\_circ\_struct\_par, 95
  - energy\_of\_circ\_structure, 94
  - energy\_of\_struct\_par, 94
  - energy\_of\_struct\_pt\_par, 96
  - energy\_of\_structure, 94
  - energy\_of\_structure\_pt, 95
- exp\_E\_Hairpin
  - loop\_energies.h, 195
- exp\_E\_IntLoop
  - loop\_energies.h, 196
- exp\_E\_Stem
  - loop\_energies.h, 195
- expand\_Full
  - RNAstruct.h, 226
- expand\_Shapiro
  - RNAstruct.h, 226
- expHairpinEnergy
  - part\_func.h, 207
- expLoopEnergy
  - part\_func.h, 207
- export\_ali\_bppm
  - consensus\_pf\_fold, 74
- export\_bppm
  - pf\_fold, 42
- export\_co\_bppm
  - pf\_cofold, 60
- export\_cofold\_arrays
  - mfe\_cofold, 56
- export\_cofold\_arrays\_gq
  - mfe\_cofold, 56
- FILENAME\_ID\_LENGTH
  - utils.h, 240
- FILENAME\_MAX\_LENGTH
  - utils.h, 240
- final\_cost
  - inverse\_fold, 98
- find\_saddle
  - findpath.h, 171
- findpath.h
  - find\_saddle, 171
  - free\_path, 172
  - get\_path, 172
- fold
  - mfe\_fold, 35
- fold.h
  - assign\_plist\_from\_db, 177
  - energy\_of\_circ\_struct, 179
  - energy\_of\_move, 175
  - energy\_of\_move\_pt, 176
  - energy\_of\_struct, 177
  - energy\_of\_struct\_pt, 178
  - HairpinE, 177
  - initialize\_fold, 177
  - loop\_energy, 176
  - LoopEnergy, 177
  - parenthesis\_structure, 175
  - parenthesis\_zucker, 175
- fold\_par

- mfe\_fold, 34
- fold\_vars.h
  - backtrack\_type, 185
  - base\_pair, 184
  - cut\_point, 183
  - dangles, 182
  - do\_backtrack, 184
  - energy\_set, 183
  - iindx, 184
  - james\_rule, 183
  - logML, 183
  - noLonelyPairs, 182
  - nonstandards, 183
  - oldAliEn, 183
  - pf\_scale, 184
  - pr, 184
  - ribo, 183
  - RibosumFile, 183
  - set\_model\_details, 182
  - temperature, 183
  - tetra\_loop, 183
- folden, 120
- FORBIDDEN
  - energy\_const.h, 169
- free\_path
  - findpath.h, 172
- free\_pf\_arrays
  - pf\_fold, 41
- free\_profile
  - profiledist.h, 217
- free\_sequence\_arrays
  - consensus\_fold, 69
- free\_tree
  - treedist.h, 233
- GASCONST
  - energy\_const.h, 169
- get\_alipf\_arrays
  - consensus\_fold, 69
- get\_boltzmann\_factor\_copy
  - energy\_parameters, 84
- get\_boltzmann\_factors
  - energy\_parameters, 84
- get\_centroid\_struct\_pl
  - centroid\_fold, 46
- get\_centroid\_struct\_pr
  - centroid\_fold, 46
- get\_concentrations
  - pf\_cofold, 61
- get\_gquad\_matrix
  - gquad.h, 186
- get\_iindx
  - utils.h, 248
- get\_indx
  - utils.h, 248
- get\_input\_line
  - utils.h, 243
- get\_line
  - utils.h, 243
- get\_monomere\_mfes
  - cofold.h, 160
- get\_mpi
  - consensus\_fold, 67
- get\_path
  - findpath.h, 172
- get\_pf\_arrays
  - pf\_fold, 43
- get\_plist
  - part\_func\_co.h, 209
- get\_scaled\_parameters
  - energy\_parameters, 83
- get\_scaled\_pf\_parameters
  - energy\_parameters, 83
- get\_TwoDfold\_variables
  - kl\_neighborhood\_mfe, 102
- get\_TwoDpfold\_variables
  - kl\_neighborhood\_pf, 105
- get\_TwoDpfold\_variables\_from\_MFE
  - kl\_neighborhood\_pf, 105
- give\_up
  - inverse\_fold, 98
- gmlRNA
  - PS\_dot.h, 220
- gquad.h
  - backtrack\_GQuad\_IntLoop, 187
  - backtrack\_GQuad\_IntLoop\_L, 187
  - get\_gquad\_matrix, 186
  - parse\_gquad, 187
- HairpinE
  - fold.h, 177
- hamming
  - utils.h, 242
- hamming\_bound
  - utils.h, 242
- iindx
  - fold\_vars.h, 184
- INF
  - energy\_const.h, 169

- init\_co\_pf\_fold
  - part\_func\_co.h, 209
- init\_pf\_fold
  - part\_func.h, 207
- init\_pf\_foldLP
  - LPfold.h, 199
- initialize\_cofold
  - cofold.h, 160
- initialize\_fold
  - fold.h, 177
- int\_urn
  - utils.h, 241
- interact, 121
- intermediate\_t, 122
- INTERVAL, 123
- inv\_verbose
  - inverse\_fold, 98
- inverse\_fold
  - final\_cost, 98
  - give\_up, 98
  - inv\_verbose, 98
  - inverse\_fold, 97
  - inverse\_pf\_fold, 98
  - inverse\_fold, 97
- inverse\_pf\_fold
  - inverse\_fold, 98
- james\_rule
  - fold\_vars.h, 183
- K0
  - energy\_const.h, 169
- kl\_neighborhood\_mfe
  - destroy\_TwoDfold\_variables, 102
  - get\_TwoDfold\_variables, 102
  - TwoDfold\_backtrack\_f5, 103
  - TwoDfoldList, 102
- kl\_neighborhood\_pf
  - destroy\_TwoDpfold\_variables, 105
  - get\_TwoDpfold\_variables, 105
  - get\_TwoDpfold\_variables\_from\_-  
MFE, 105
  - TwoDpfoldList, 106
- kl\_neighborhood\_stochbt
  - TwoDpfold\_pbacktrack, 107
  - TwoDpfold\_pbacktrack5, 108
- Lfold
  - local\_mfe\_fold, 77
- Lfoldz
  - local\_mfe\_fold, 77
- LIST, 124
- Local MFE consensus structures for Sequence Alignments, 81
- Local MFE structure Prediction and Z-scores, 77
- local\_consensus\_fold
  - aliLfold, 81
- local\_mfe\_fold
  - Lfold, 77
  - Lfoldz, 77
- local\_pf\_fold
  - pfl\_fold, 78
  - putoutU\_prob, 79
  - putoutU\_prob\_bin, 79
  - update\_pf\_paramsLP, 78
- logML
  - fold\_vars.h, 183
- loop\_energies.h
  - E\_Hairpin, 193
  - E\_IntLoop, 192
  - E\_Stem, 194
  - exp\_E\_Hairpin, 195
  - exp\_E\_IntLoop, 196
  - exp\_E\_Stem, 195
- loop\_energy
  - fold.h, 176
- LoopEnergy
  - fold.h, 177
- LPfold.h
  - init\_pf\_foldLP, 199
- LST\_BUCKET, 125
- Make\_bp\_profile
  - profiledist.h, 217
- Make\_bp\_profile\_bppm
  - profiledist.h, 216
- make\_loop\_index\_pt
  - utils.h, 246
- make\_pair\_table
  - utils.h, 246
- make\_pair\_table\_snoop
  - utils.h, 246
- Make\_swString
  - stringdist.h, 229
- make\_tree
  - treedist.h, 232
- MAX2
  - utils.h, 239
- MAX3

- utils.h, 240
- MAXLOOP
  - energy\_const.h, 170
- MEA
  - MEA.h, 200
- MEA.h
  - MEA, 200
- mean\_bp\_dist
  - part\_func.h, 207
- mean\_bp\_distance
  - pf\_fold, 43
- mean\_bp\_distance\_pr
  - pf\_fold, 43
- MFE Consensus Structures for Sequence Alignment(s), 71
- MFE Structures of two hybridized Sequences, 55
- mfe\_cofold
  - cofold, 55
  - export\_cofold\_arrays, 56
  - export\_cofold\_arrays\_gq, 56
- mfe\_fold
  - circfold, 35
  - fold, 35
  - fold\_par, 34
- MIN2
  - utils.h, 239
- MIN3
  - utils.h, 239
- model\_detailsT, 126
  - dangles, 126
- move\_t, 128
- NBPAIRS
  - energy\_const.h, 170
- nc\_fact
  - consensus\_fold, 70
- noLonelyPairs
  - fold\_vars.h, 182
- nonstandards
  - fold\_vars.h, 183
- nerror
  - utils.h, 241
- oldAliEn
  - fold\_vars.h, 183
- pack\_structure
  - utils.h, 245
- PAIR, 129
- pair\_info, 130
- pairpro, 131
- paramT, 132
- parenthesis\_structure
  - fold.h, 175
- parenthesis\_zuker
  - fold.h, 175
- parse\_gquad
  - gquad.h, 187
- parse\_structure
  - RNAstruct.h, 227
- Parsing and Comparing - Functions to Manipulate Structures, 110
- part\_func.h
  - centroid, 207
  - expHairpinEnergy, 207
  - expLoopEnergy, 207
  - init\_pf\_fold, 207
  - mean\_bp\_dist, 207
- part\_func\_co.h
  - get\_plist, 209
  - init\_co\_pf\_fold, 209
- Partition Function and Base Pair Probabilities for Sequence Alignment(s), 73
- Partition Function for two hybridized Sequences, 58
- Partition Function for two hybridized Sequences as a stepwise Process, 63
- Partition functions for locally stable secondary structures, 78
- path\_t, 133
- pbacktrack
  - subopt\_stochbt, 52
- pbacktrack\_circ
  - subopt\_stochbt, 53
- pf\_circ\_fold
  - pf\_fold, 40
- pf\_cofold
  - co\_pf\_fold, 59
  - co\_pf\_fold\_par, 59
  - compute\_probabilities, 61
  - export\_co\_bppm, 60
  - get\_concentrations, 61
  - update\_co\_pf\_params, 60
  - update\_co\_pf\_params\_par, 61
- pf\_fold
  - assign\_plist\_from\_pr, 42
  - export\_bppm, 42

- free\_pf\_arrays, 41
- get\_pf\_arrays, 43
- mean\_bp\_distance, 43
- mean\_bp\_distance\_pr, 43
- pf\_circ\_fold, 40
- pf\_fold, 39
- pf\_fold\_par, 39
- pf\_fold, 39
- update\_pf\_params, 42
- pf\_fold\_par
  - pf\_fold, 39
- pf\_interact
  - up\_cofold, 64
- pf\_paramT, 134
  - alpha, 134
- pf\_scale
  - fold\_vars.h, 184
- pf\_unstru
  - up\_cofold, 63
- pfl\_fold
  - local\_pf\_fold, 78
- plist, 136
- plot\_layouts.h
  - rna\_plot\_type, 215
  - simple\_circplot\_coordinates, 214
  - simple\_xy\_coordinates, 214
  - VRNA\_PLOT\_TYPE\_CIRCULAR, 213
  - VRNA\_PLOT\_TYPE\_NAVIEW, 213
  - VRNA\_PLOT\_TYPE\_SIMPLE, 213
- Postorder\_list, 137
- pr
  - fold\_vars.h, 184
- Predicting Consensus Structures from Alignment(s), 66
- Predicting Locally stable structures of large sequences, 76
- print\_tty\_constraint
  - utils.h, 247
- print\_tty\_input\_seq
  - utils.h, 247
- print\_tty\_input\_seq\_str
  - utils.h, 247
- profile\_edit\_distance
  - profiledist.h, 216
- profiledist.h
  - free\_profile, 217
  - Make\_bp\_profile, 217
  - Make\_bp\_profile\_bppm, 216
  - profile\_edit\_distance, 216
- PS\_dot.h
  - aliPS\_color\_aln, 221
  - gmlRNA, 220
  - PS\_dot\_plot, 222
  - PS\_dot\_plot\_list, 221
  - PS\_rna\_plot, 219
  - PS\_rna\_plot\_a, 219
  - ssv\_rna\_plot, 220
  - svg\_rna\_plot, 220
  - xrna\_plot, 221
- PS\_dot\_plot
  - PS\_dot.h, 222
- PS\_dot\_plot\_list
  - PS\_dot.h, 221
- PS\_rna\_plot
  - PS\_dot.h, 219
- PS\_rna\_plot\_a
  - PS\_dot.h, 219
- pu\_contrib, 138
- pu\_out, 139
- putoutpU\_prob
  - local\_pf\_fold, 79
- putoutpU\_prob\_bin
  - local\_pf\_fold, 79
- random\_string
  - utils.h, 242
- read\_parameter\_file
  - energy\_parameters\_rw, 86
- read\_record
  - utils.h, 244
- Reading/Writing energy parameter sets from/to File, 86
- ribo
  - fold\_vars.h, 183
- RibosumFile
  - fold\_vars.h, 183
- RNA Secondary Structure Folding, 31
- rna\_plot\_type
  - plot\_layouts.h, 215
- RNAstruct.h
  - add\_root, 226
  - b2C, 225
  - b2HIT, 225
  - b2Shapiro, 226
  - expand\_Full, 226
  - expand\_Shapiro, 226
  - parse\_structure, 227

- unexpand\_aligned\_F, 227
- unexpand\_Full, 227
- unweight, 227
- scale\_parameters
  - energy\_parameters, 83
- Searching Sequences for Predefined Structures, 97
- sect, 140
- set\_model\_details
  - fold\_vars.h, 182
- simple\_circplot\_coordinates
  - plot\_layouts.h, 214
- simple\_xy\_coordinates
  - plot\_layouts.h, 214
- snoopT, 141
- SOLUTION, 142
- space
  - utils.h, 240
- ssv\_rna\_plot
  - PS\_dot.h, 220
- st\_back
  - subopt\_stochbt, 53
- Stochastic backtracking in the Ensemble, 52
- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), 75
- Stochastic Backtracking of Structures from Distance Based Partitioning, 107
- STR
  - utils.h, 240
- str\_DNA2RNA
  - utils.h, 247
- str\_uppercase
  - utils.h, 247
- string\_edit\_distance
  - stringdist.h, 229
- stringdist.h
  - Make\_swString, 229
  - string\_edit\_distance, 229
- struct\_en, 143
- subopt
  - subopt\_wuchty, 50
- subopt\_circ
  - subopt\_wuchty, 51
- subopt\_stochbt
  - pbacktrack, 52
  - pbacktrack\_circ, 53
- st\_back, 53
- subopt\_wuchty
  - subopt, 50
  - subopt\_circ, 51
- subopt\_zuker
  - zukersubopt, 49
- Suboptimal structures according to Zuker et al. 1989, 49
- Suboptimal structures within an energy band around the MFE, 50
- svg\_rna\_plot
  - PS\_dot.h, 220
- svm\_model, 144
- swString, 145
- temperature
  - fold\_vars.h, 183
- tetra\_loop
  - fold\_vars.h, 183
- time\_stamp
  - utils.h, 242
- Tree, 146
- tree\_edit\_distance
  - treedist.h, 232
- treedist.h
  - free\_tree, 233
  - make\_tree, 232
  - tree\_edit\_distance, 232
- TURN
  - energy\_const.h, 170
- TwoDfold\_backtrack\_f5
  - kl\_neighborhood\_mfe, 103
- TwoDfold\_solution, 147
- TwoDfold\_vars, 148
- TwoDfoldList
  - kl\_neighborhood\_mfe, 102
- TwoDpfold\_pbacktrack
  - kl\_neighborhood\_stochbt, 107
- TwoDpfold\_pbacktrack5
  - kl\_neighborhood\_stochbt, 108
- TwoDpfold\_solution, 150
- TwoDpfold\_vars, 151
- TwoDpfoldList
  - kl\_neighborhood\_pf, 106
- unexpand\_aligned\_F
  - RNAstruct.h, 227
- unexpand\_Full
  - RNAstruct.h, 227
- unpack\_structure



- utils.h, 245
- unweight
  - RNAstruct.h, 227
- up\_cofold
  - pf\_interact, 64
  - pf\_unstru, 63
- update\_alifold\_params
  - alifold.h, 158
- update\_co\_pf\_params
  - pf\_cofold, 60
- update\_co\_pf\_params\_par
  - pf\_cofold, 61
- update\_pf\_params
  - pf\_fold, 42
- update\_pf\_paramsLP
  - local\_pf\_fold, 78
- urn
  - utils.h, 241
- utils.h
  - alimake\_pair\_table, 246
  - constrain\_ptypes, 248
  - copy\_pair\_table, 246
  - FILENAME\_ID\_LENGTH, 240
  - FILENAME\_MAX\_LENGTH, 240
  - get\_iindx, 248
  - get\_indx, 248
  - get\_input\_line, 243
  - get\_line, 243
  - hamming, 242
  - hamming\_bound, 242
  - int\_urn, 241
  - make\_loop\_index\_pt, 246
  - make\_pair\_table, 246
  - make\_pair\_table\_snoop, 246
  - MAX2, 239
  - MAX3, 240
  - MIN2, 239
  - MIN3, 239
  - nerror, 241
  - pack\_structure, 245
  - print\_tty\_constraint, 247
  - print\_tty\_input\_seq, 247
  - print\_tty\_input\_seq\_str, 247
  - random\_string, 242
  - read\_record, 244
  - space, 240
  - STR, 240
  - str\_DNA2RNA, 247
  - str\_uppercase, 247
  - time\_stamp, 242
  - unpack\_structure, 245
  - urn, 241
  - VRNA\_CONSTRAINT\_ALL, 239
  - VRNA\_CONSTRAINT\_ANG\_-  
BRACK, 238
  - VRNA\_CONSTRAINT\_DOT, 238
  - VRNA\_CONSTRAINT\_G, 239
  - VRNA\_CONSTRAINT\_-  
MULTILINE, 239
  - VRNA\_CONSTRAINT\_NO\_-  
HEADER, 239
  - VRNA\_CONSTRAINT\_PIPE, 238
  - VRNA\_CONSTRAINT\_RND\_-  
BRACK, 239
  - VRNA\_CONSTRAINT\_X, 238
  - VRNA\_INPUT\_BLANK\_LINE,  
238
  - VRNA\_INPUT\_COMMENT, 238
  - VRNA\_INPUT\_CONSTRAINT,  
237
  - VRNA\_INPUT\_ERROR, 237
  - VRNA\_INPUT\_FASTA\_HEADER,  
237
  - VRNA\_INPUT\_MISC, 237
  - VRNA\_INPUT\_NO\_REST, 238
  - VRNA\_INPUT\_NO\_SPAN, 238
  - VRNA\_INPUT\_NO\_-  
TRUNCATION, 237
  - VRNA\_INPUT\_NOSKIP\_-  
BLANK\_LINES, 238
  - VRNA\_INPUT\_NOSKIP\_-  
COMMENTS, 238
  - VRNA\_INPUT\_QUIT, 237
  - VRNA\_INPUT\_SEQUENCE, 237
  - VRNA\_OPTION\_MULTILINE,  
239
  - warn\_user, 241
  - xrealloc, 240
  - XSTR, 240
  - xsubi, 249
  - VRNA\_CONSTRAINT\_ALL  
utils.h, 239
  - VRNA\_CONSTRAINT\_ANG\_BRACK  
utils.h, 238
  - VRNA\_CONSTRAINT\_DOT  
utils.h, 238
  - VRNA\_CONSTRAINT\_G  
utils.h, 239
  - VRNA\_CONSTRAINT\_MULTILINE

- utils.h, 239
- VRNA\_CONSTRAINT\_NO\_HEADER
  - utils.h, 239
- VRNA\_CONSTRAINT\_PIPE
  - utils.h, 238
- VRNA\_CONSTRAINT\_RND\_BRACK
  - utils.h, 239
- VRNA\_CONSTRAINT\_X
  - utils.h, 238
- VRNA\_CONVERT\_OUTPUT\_ALL
  - energy\_parameters\_convert, 89
- VRNA\_CONVERT\_OUTPUT\_BULGE
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_-
  - DANGLE3
    - energy\_parameters\_convert, 90
  - DANGLE5
    - energy\_parameters\_convert, 89
- VRNA\_CONVERT\_OUTPUT\_DUMP
  - energy\_parameters\_convert, 91
- VRNA\_CONVERT\_OUTPUT\_HP
  - energy\_parameters\_convert, 89
- VRNA\_CONVERT\_OUTPUT\_INT
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_INT\_11
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_INT\_21
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_INT\_22
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_MISC
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_ML
  - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_MM\_-
  - EXT
    - energy\_parameters\_convert, 89
  - VRNA\_CONVERT\_OUTPUT\_MM\_HP
    - energy\_parameters\_convert, 89
  - VRNA\_CONVERT\_OUTPUT\_MM\_-
    - INT
      - energy\_parameters\_convert, 89
    - VRNA\_CONVERT\_OUTPUT\_MM\_-
      - INT\_1N
        - energy\_parameters\_convert, 89
      - VRNA\_CONVERT\_OUTPUT\_MM\_-
        - INT\_23
          - energy\_parameters\_convert, 89
- VRNA\_CONVERT\_OUTPUT\_MM\_-
  - MULTI
    - energy\_parameters\_convert, 89
- VRNA\_CONVERT\_OUTPUT\_NINIO
  - energy\_parameters\_convert, 91
- VRNA\_CONVERT\_OUTPUT\_-
  - SPECIAL\_HP
    - energy\_parameters\_convert, 90
- VRNA\_CONVERT\_OUTPUT\_STACK
  - energy\_parameters\_convert, 89
- VRNA\_CONVERT\_OUTPUT\_-
  - VANILLA
    - energy\_parameters\_convert, 90
- VRNA\_INPUT\_BLANK\_LINE
  - utils.h, 238
- VRNA\_INPUT\_COMMENT
  - utils.h, 238
- VRNA\_INPUT\_CONSTRAINT
  - utils.h, 237
- VRNA\_INPUT\_ERROR
  - utils.h, 237
- VRNA\_INPUT\_FASTA\_HEADER
  - utils.h, 237
- VRNA\_INPUT\_MISC
  - utils.h, 237
- VRNA\_INPUT\_NO\_REST
  - utils.h, 238
- VRNA\_INPUT\_NO\_SPAN
  - utils.h, 238
- VRNA\_INPUT\_NO\_TRUNCATION
  - utils.h, 237
- VRNA\_INPUT\_NOSKIP\_BLANK\_-
  - LINES
    - utils.h, 238
- VRNA\_INPUT\_NOSKIP\_COMMENTS
  - utils.h, 238
- VRNA\_INPUT\_QUIT
  - utils.h, 237
- VRNA\_INPUT\_SEQUENCE
  - utils.h, 237
- VRNA\_OPTION\_MULTILINE
  - utils.h, 239
- VRNA\_PLOT\_TYPE\_CIRCULAR
  - plot\_layouts.h, 213
- VRNA\_PLOT\_TYPE\_NAVIEW
  - plot\_layouts.h, 213
- VRNA\_PLOT\_TYPE\_SIMPLE
  - plot\_layouts.h, 213
- warn\_user

---

utils.h, [241](#)  
write\_parameter\_file  
  energy\_parameters\_rw, [86](#)  
  
xrealloc  
  utils.h, [240](#)  
xrna\_plot  
  PS\_dot.h, [221](#)  
XSTR  
  utils.h, [240](#)  
xsubi  
  utils.h, [249](#)  
  
zukersubopt  
  subopt\_zuker, [49](#)