

RNAlib-2.1.9

Generated by Doxygen 1.8.3.1

Thu Feb 5 2015 15:05:42

Contents

1	ViennaRNA Package core - RNAlib	1
1.1	Introduction	1
2	Parsing and Comparing - Functions to Manipulate Structures	3
3	Utilities - Odds and Ends	5
3.1	Producing secondary structure graphs	5
4	Example - A Small Example Program	7
5	Deprecated List	9
6	Module Index	11
6.1	Modules	11
7	Data Structure Index	13
7.1	Data Structures	13
8	File Index	15
8.1	File List	15
9	Module Documentation	17
9.1	RNA Secondary Structure Folding	17
9.1.1	Detailed Description	19
9.2	Calculating Minimum Free Energy (MFE) Structures	20
9.2.1	Detailed Description	20
9.2.2	Function Documentation	21
9.2.2.1	fold_par	21
9.2.2.2	fold	22
9.2.2.3	circfold	22
9.3	Calculating Partition Functions and Pair Probabilities	23
9.3.1	Detailed Description	24
9.3.2	Function Documentation	24
9.3.2.1	pf_fold_par	24

9.3.2.2	pf_fold	25
9.3.2.3	pf_circ_fold	26
9.3.2.4	free_pf_arrays	27
9.3.2.5	update_pf_params	27
9.3.2.6	export_bppm	27
9.3.2.7	assign_plist_from_pr	28
9.3.2.8	get_pf_arrays	28
9.3.2.9	mean_bp_distance	28
9.3.2.10	mean_bp_distance_pr	29
9.4	Compute the structure with maximum expected accuracy (MEA)	30
9.5	Compute the centroid structure	31
9.5.1	Detailed Description	31
9.5.2	Function Documentation	31
9.5.2.1	get_centroid_struct_pl	31
9.5.2.2	get_centroid_struct_pr	31
9.6	Enumerating Suboptimal Structures	33
9.6.1	Detailed Description	33
9.7	Suboptimal structures according to Zuker et al. 1989	34
9.7.1	Detailed Description	34
9.7.2	Function Documentation	34
9.7.2.1	zukersubopt	34
9.8	Suboptimal structures within an energy band around the MFE	35
9.8.1	Detailed Description	35
9.8.2	Function Documentation	35
9.8.2.1	subopt	35
9.8.2.2	subopt_circ	36
9.9	Stochastic backtracking in the Ensemble	37
9.9.1	Detailed Description	37
9.9.2	Function Documentation	37
9.9.2.1	pbacktrack	37
9.9.2.2	pbacktrack_circ	38
9.9.3	Variable Documentation	38
9.9.3.1	st_back	38
9.10	Calculate Secondary Structures of two RNAs upon Dimerization	39
9.10.1	Detailed Description	39
9.11	MFE Structures of two hybridized Sequences	40
9.11.1	Detailed Description	40
9.11.2	Function Documentation	40
9.11.2.1	cofold	40
9.11.2.2	export_cofold_arrays_gq	41

9.11.2.3	export_cofold_arrays	41
9.12	Partition Function for two hybridized Sequences	42
9.12.1	Detailed Description	43
9.12.2	Function Documentation	43
9.12.2.1	co_pf_fold	43
9.12.2.2	co_pf_fold_par	43
9.12.2.3	export_co_bppm	44
9.12.2.4	update_co_pf_params	44
9.12.2.5	update_co_pf_params_par	44
9.12.2.6	compute_probabilities	45
9.12.2.7	get_concentrations	45
9.13	Partition Function for two hybridized Sequences as a stepwise Process	46
9.13.1	Detailed Description	46
9.13.2	Function Documentation	46
9.13.2.1	pf_unstru	46
9.13.2.2	pf_interact	47
9.14	Predicting Consensus Structures from Alignment(s)	48
9.14.1	Detailed Description	49
9.14.2	Function Documentation	49
9.14.2.1	get_mpi	49
9.14.2.2	energy_of_alistruct	49
9.14.2.3	encode_ali_sequence	50
9.14.2.4	alloc_sequence_arrays	50
9.14.2.5	free_sequence_arrays	50
9.14.2.6	get_alipf_arrays	51
9.14.3	Variable Documentation	51
9.14.3.1	cv_fact	51
9.14.3.2	nc_fact	51
9.15	MFE Consensus Structures for Sequence Alignment(s)	52
9.15.1	Detailed Description	52
9.15.2	Function Documentation	52
9.15.2.1	alifold	52
9.15.2.2	circularfold	52
9.16	Partition Function and Base Pair Probabilities for Sequence Alignment(s)	54
9.16.1	Detailed Description	54
9.16.2	Function Documentation	54
9.16.2.1	alipf_fold_par	54
9.16.2.2	alipf_fold	54
9.16.2.3	alipf_circ_fold	55
9.16.2.4	export_ali_bppm	55

9.17 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	56
9.17.1 Detailed Description	56
9.17.2 Function Documentation	56
9.17.2.1 alipbacktrack	56
9.18 Predicting Locally stable structures of large sequences	57
9.18.1 Detailed Description	57
9.19 Local MFE structure Prediction and Z-scores	58
9.19.1 Detailed Description	58
9.19.2 Function Documentation	58
9.19.2.1 Lfold	58
9.19.2.2 Lfoldz	58
9.20 Partition functions for locally stable secondary structures	59
9.20.1 Detailed Description	59
9.20.2 Function Documentation	59
9.20.2.1 update_pf_paramsLP	59
9.20.2.2 pfl_fold	59
9.20.2.3 putoutU_prob	60
9.20.2.4 putoutU_prob_bin	60
9.21 Local MFE consensus structures for Sequence Alignments	61
9.21.1 Detailed Description	61
9.21.2 Function Documentation	61
9.21.2.1 aliLfold	61
9.22 Change and Precalculate Energy Parameter Sets and Boltzmann Factors	62
9.22.1 Detailed Description	62
9.22.2 Function Documentation	63
9.22.2.1 scale_parameters	63
9.22.2.2 get_scaled_parameters	63
9.22.2.3 get_scaled_pf_parameters	63
9.22.2.4 get_boltzmann_factors	63
9.22.2.5 get_boltzmann_factor_copy	64
9.23 Reading/Writing energy parameter sets from/to File	65
9.23.1 Detailed Description	65
9.23.2 Function Documentation	65
9.23.2.1 read_parameter_file	65
9.23.2.2 write_parameter_file	65
9.24 Converting energy parameter files	67
9.24.1 Detailed Description	67
9.24.2 Macro Definition Documentation	68
9.24.2.1 VRNA_CONVERT_OUTPUT_ALL	68
9.24.2.2 VRNA_CONVERT_OUTPUT_HP	68

9.24.2.3	VRNA_CONVERT_OUTPUT_STACK	68
9.24.2.4	VRNA_CONVERT_OUTPUT_MM_HP	68
9.24.2.5	VRNA_CONVERT_OUTPUT_MM_INT	68
9.24.2.6	VRNA_CONVERT_OUTPUT_MM_INT_1N	68
9.24.2.7	VRNA_CONVERT_OUTPUT_MM_INT_23	68
9.24.2.8	VRNA_CONVERT_OUTPUT_MM_MULTI	68
9.24.2.9	VRNA_CONVERT_OUTPUT_MM_EXT	68
9.24.2.10	VRNA_CONVERT_OUTPUT_DANGLE5	68
9.24.2.11	VRNA_CONVERT_OUTPUT_DANGLE3	68
9.24.2.12	VRNA_CONVERT_OUTPUT_INT_11	68
9.24.2.13	VRNA_CONVERT_OUTPUT_INT_21	69
9.24.2.14	VRNA_CONVERT_OUTPUT_INT_22	69
9.24.2.15	VRNA_CONVERT_OUTPUT_BULGE	69
9.24.2.16	VRNA_CONVERT_OUTPUT_INT	69
9.24.2.17	VRNA_CONVERT_OUTPUT_ML	69
9.24.2.18	VRNA_CONVERT_OUTPUT_MISC	69
9.24.2.19	VRNA_CONVERT_OUTPUT_SPECIAL_HP	69
9.24.2.20	VRNA_CONVERT_OUTPUT_VANILLA	69
9.24.2.21	VRNA_CONVERT_OUTPUT_NINIO	69
9.24.2.22	VRNA_CONVERT_OUTPUT_DUMP	69
9.24.3	Function Documentation	69
9.24.3.1	convert_parameter_file	69
9.25	Energy evaluation	71
9.25.1	Detailed Description	71
9.25.2	Function Documentation	71
9.25.2.1	energy_of_structure	71
9.25.2.2	energy_of_struct_par	72
9.25.2.3	energy_of_circ_structure	72
9.25.2.4	energy_of_circ_struct_par	73
9.25.2.5	energy_of_structure_pt	73
9.25.2.6	energy_of_struct_pt_par	74
9.26	Searching Sequences for Predefined Structures	75
9.26.1	Detailed Description	75
9.26.2	Function Documentation	75
9.26.2.1	inverse_fold	75
9.26.2.2	inverse_pf_fold	76
9.26.3	Variable Documentation	76
9.26.3.1	final_cost	76
9.26.3.2	give_up	76
9.26.3.3	inv_verbose	76

9.27	Classified Dynamic Programming	77
9.27.1	Detailed Description	77
9.28	Distance based partitioning of the Secondary Structure Space	78
9.28.1	Detailed Description	78
9.29	Calculating MFE representatives of a Distance Based Partitioning	79
9.29.1	Detailed Description	79
9.29.2	Function Documentation	79
9.29.2.1	get_TwoDfold_variables	79
9.29.2.2	destroy_TwoDfold_variables	80
9.29.2.3	TwoDfoldList	80
9.29.2.4	TwoDfold_backtrack_f5	81
9.30	Calculate Partition Functions of a Distance Based Partitioning	82
9.30.1	Detailed Description	82
9.30.2	Function Documentation	82
9.30.2.1	get_TwoDpfold_variables	82
9.30.2.2	get_TwoDpfold_variables_from_MFE	83
9.30.2.3	destroy_TwoDpfold_variables	83
9.30.2.4	TwoDpfoldList	83
9.31	Stochastic Backtracking of Structures from Distance Based Partitioning	85
9.31.1	Detailed Description	85
9.31.2	Function Documentation	85
9.31.2.1	TwoDpfold_pbacktrack	85
9.31.2.2	TwoDpfold_pbacktrack5	86
9.32	Compute the Density of States	87
9.32.1	Detailed Description	87
9.32.2	Variable Documentation	87
9.32.2.1	density_of_states	87
9.33	Parsing and Comparing - Functions to Manipulate Structures	88
10	Data Structure Documentation	89
10.1	bondT Struct Reference	89
10.1.1	Detailed Description	89
10.2	bondTEn Struct Reference	89
10.2.1	Detailed Description	89
10.3	cofoldF Struct Reference	89
10.4	ConcEnt Struct Reference	90
10.5	constrain Struct Reference	90
10.5.1	Detailed Description	90
10.6	COORDINATE Struct Reference	90
10.6.1	Detailed Description	90

10.7 cpair Struct Reference	90
10.7.1 Detailed Description	91
10.8 duplexT Struct Reference	91
10.9 dupVar Struct Reference	91
10.10folden Struct Reference	91
10.11interact Struct Reference	91
10.12intermediate_t Struct Reference	92
10.13INTERVAL Struct Reference	92
10.13.1 Detailed Description	92
10.14LIST Struct Reference	93
10.15LST_BUCKET Struct Reference	93
10.16model_detailsT Struct Reference	93
10.16.1 Detailed Description	94
10.16.2 Field Documentation	94
10.16.2.1 dangles	94
10.17move_t Struct Reference	94
10.18PAIR Struct Reference	94
10.18.1 Detailed Description	94
10.19pair_info Struct Reference	95
10.19.1 Detailed Description	95
10.20pairpro Struct Reference	96
10.21paramT Struct Reference	96
10.21.1 Detailed Description	97
10.22path_t Struct Reference	97
10.23pf_paramT Struct Reference	97
10.23.1 Detailed Description	97
10.23.2 Field Documentation	98
10.23.2.1 alpha	98
10.24plist Struct Reference	98
10.24.1 Detailed Description	98
10.25Postorder_list Struct Reference	98
10.26pu_contrib Struct Reference	98
10.26.1 Detailed Description	99
10.27pu_out Struct Reference	99
10.27.1 Detailed Description	99
10.28sect Struct Reference	99
10.28.1 Detailed Description	99
10.29snoopT Struct Reference	99
10.30SOLUTION Struct Reference	100
10.30.1 Detailed Description	100

10.31	struct_en Struct Reference	100
10.32	svm_model Struct Reference	100
10.33	swString Struct Reference	100
10.34	Tree Struct Reference	101
10.35	TwoDfold_solution Struct Reference	101
10.35.1	Detailed Description	101
10.36	TwoDfold_vars Struct Reference	102
10.36.1	Detailed Description	103
10.37	TwoDpfold_solution Struct Reference	103
10.37.1	Detailed Description	103
10.38	TwoDpfold_vars Struct Reference	103
10.38.1	Detailed Description	105
11	File Documentation	107
11.1	/home/mescalini/ronny/WORK/ViennaRNA/H/2Dfold.h File Reference	107
11.2	/home/mescalini/ronny/WORK/ViennaRNA/H/2Dpfold.h File Reference	108
11.3	/home/mescalini/ronny/WORK/ViennaRNA/H/alifold.h File Reference	108
11.3.1	Detailed Description	110
11.3.2	Function Documentation	110
11.3.2.1	update_alifold_params	110
11.4	/home/mescalini/ronny/WORK/ViennaRNA/H/cofold.h File Reference	110
11.4.1	Detailed Description	112
11.4.2	Function Documentation	112
11.4.2.1	get_monomere_mfes	112
11.4.2.2	initialize_cofold	112
11.5	/home/mescalini/ronny/WORK/ViennaRNA/H/convert_epars.h File Reference	112
11.5.1	Detailed Description	113
11.6	/home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h File Reference	113
11.6.1	Detailed Description	114
11.7	/home/mescalini/ronny/WORK/ViennaRNA/H/dist_vars.h File Reference	115
11.7.1	Detailed Description	115
11.7.2	Variable Documentation	115
11.7.2.1	edit_backtrack	115
11.7.2.2	cost_matrix	115
11.8	/home/mescalini/ronny/WORK/ViennaRNA/H/duplex.h File Reference	116
11.8.1	Detailed Description	116
11.9	/home/mescalini/ronny/WORK/ViennaRNA/H/edit_cost.h File Reference	116
11.9.1	Detailed Description	116
11.10	/home/mescalini/ronny/WORK/ViennaRNA/H/energy_const.h File Reference	117
11.10.1	Detailed Description	117

11.10.2 Macro Definition Documentation	117
11.10.2.1 GASCONST	117
11.10.2.2 K0	117
11.10.2.3 INF	118
11.10.2.4 FORBIDDEN	118
11.10.2.5 BONUS	118
11.10.2.6 NBPAIRS	118
11.10.2.7 TURN	118
11.10.2.8 MAXLOOP	118
11.11/home/mescalini/ronny/WORK/ViennaRNA/H/findpath.h File Reference	118
11.11.1 Detailed Description	119
11.11.2 Function Documentation	119
11.11.2.1 find_saddle	119
11.11.2.2 get_path	120
11.11.2.3 free_path	120
11.12/home/mescalini/ronny/WORK/ViennaRNA/H/fold.h File Reference	120
11.12.1 Detailed Description	122
11.12.2 Function Documentation	122
11.12.2.1 parenthesis_structure	122
11.12.2.2 parenthesis_zuker	123
11.12.2.3 energy_of_move	123
11.12.2.4 energy_of_move_pt	123
11.12.2.5 loop_energy	123
11.12.2.6 assign_plist_from_db	124
11.12.2.7 LoopEnergy	124
11.12.2.8 HairpinE	124
11.12.2.9 initialize_fold	124
11.12.2.10energy_of_struct	124
11.12.2.11energy_of_struct_pt	125
11.12.2.12energy_of_circ_struct	125
11.13/home/mescalini/ronny/WORK/ViennaRNA/H/fold_vars.h File Reference	126
11.13.1 Detailed Description	128
11.13.2 Function Documentation	128
11.13.2.1 set_model_details	128
11.13.3 Variable Documentation	128
11.13.3.1 noLonelyPairs	128
11.13.3.2 dangles	128
11.13.3.3 tetra_loop	129
11.13.3.4 energy_set	129
11.13.3.5 oldAliEn	129

11.13.3.6	ribo	129
11.13.3.7	RibosumFile	129
11.13.3.8	nonstandards	129
11.13.3.9	temperature	129
11.13.3.10	games_rule	129
11.13.3.11	logML	130
11.13.3.12	cut_point	130
11.13.3.13	base_pair	130
11.13.3.14	pr	130
11.13.3.15	indx	130
11.13.3.16	pf_scale	130
11.13.3.17	do_backtrack	130
11.13.3.18	backtrack_type	131
11.13.3.19	canonicalBPonly	131
11.14	home/mescal/ronny/WORK/ViennaRNA/H/gquad.h File Reference	131
11.14.1	Detailed Description	131
11.14.2	Function Documentation	132
11.14.2.1	get_gquad_matrix	132
11.14.2.2	parse_gquad	132
11.14.2.3	backtrack_GQuad_IntLoop	132
11.14.2.4	backtrack_GQuad_IntLoop_L	133
11.15	home/mescal/ronny/WORK/ViennaRNA/H/inverse.h File Reference	133
11.15.1	Detailed Description	133
11.16	home/mescal/ronny/WORK/ViennaRNA/H/Lfold.h File Reference	133
11.16.1	Detailed Description	134
11.17	home/mescal/ronny/WORK/ViennaRNA/H/loop_energies.h File Reference	134
11.17.1	Detailed Description	134
11.17.2	Function Documentation	135
11.17.2.1	E_IntLoop	135
11.17.2.2	E_Hairpin	136
11.17.2.3	E_Stem	136
11.17.2.4	exp_E_Stem	137
11.17.2.5	exp_E_Hairpin	138
11.17.2.6	exp_E_IntLoop	138
11.18	home/mescal/ronny/WORK/ViennaRNA/H/LPfold.h File Reference	139
11.18.1	Detailed Description	140
11.18.2	Function Documentation	140
11.18.2.1	init_pf_foldLP	140
11.19	home/mescal/ronny/WORK/ViennaRNA/H/MEA.h File Reference	140
11.19.1	Detailed Description	141

11.19.2 Function Documentation	141
11.19.2.1 MEA	141
11.20/home/mescalini/ronny/WORK/ViennaRNA/H/mm.h File Reference	141
11.20.1 Detailed Description	141
11.21/home/mescalini/ronny/WORK/ViennaRNA/H/naview.h File Reference	141
11.22/home/mescalini/ronny/WORK/ViennaRNA/H/params.h File Reference	142
11.23/home/mescalini/ronny/WORK/ViennaRNA/H/part_func.h File Reference	143
11.23.1 Detailed Description	144
11.23.2 Function Documentation	145
11.23.2.1 init_pf_fold	145
11.23.2.2 centroid	145
11.23.2.3 mean_bp_dist	145
11.23.2.4 expLoopEnergy	145
11.23.2.5 expHairpinEnergy	145
11.24/home/mescalini/ronny/WORK/ViennaRNA/H/part_func_co.h File Reference	145
11.24.1 Detailed Description	147
11.24.2 Function Documentation	147
11.24.2.1 get_plist	147
11.24.2.2 init_co_pf_fold	147
11.25/home/mescalini/ronny/WORK/ViennaRNA/H/part_func_up.h File Reference	147
11.25.1 Detailed Description	148
11.26/home/mescalini/ronny/WORK/ViennaRNA/H/plot_layouts.h File Reference	148
11.26.1 Detailed Description	150
11.26.2 Macro Definition Documentation	150
11.26.2.1 VRNA_PLOT_TYPE_SIMPLE	150
11.26.2.2 VRNA_PLOT_TYPE_NAVIEW	150
11.26.2.3 VRNA_PLOT_TYPE_CIRCULAR	150
11.26.3 Function Documentation	151
11.26.3.1 simple_xy_coordinates	151
11.26.3.2 simple_circplot_coordinates	151
11.26.4 Variable Documentation	151
11.26.4.1 rna_plot_type	151
11.27/home/mescalini/ronny/WORK/ViennaRNA/H/profiledist.h File Reference	152
11.27.1 Function Documentation	153
11.27.1.1 profile_edit_distance	153
11.27.1.2 Make_bp_profile_bppm	153
11.27.1.3 free_profile	153
11.27.1.4 Make_bp_profile	153
11.28/home/mescalini/ronny/WORK/ViennaRNA/H/PS_dot.h File Reference	153
11.28.1 Detailed Description	154

11.28.2 Function Documentation	155
11.28.2.1 PS_rna_plot	155
11.28.2.2 PS_rna_plot_a	155
11.28.2.3 gmlRNA	155
11.28.2.4 ssv_rna_plot	156
11.28.2.5 svg_rna_plot	156
11.28.2.6 xrna_plot	156
11.28.2.7 PS_dot_plot_list	156
11.28.2.8 aliPS_color_aln	157
11.28.2.9 PS_dot_plot	157
11.29/home/mescal/ronny/WORK/ViennaRNA/H/read_epars.h File Reference	157
11.30/home/mescal/ronny/WORK/ViennaRNA/H/RNAstruct.h File Reference	157
11.30.1 Detailed Description	158
11.30.2 Function Documentation	159
11.30.2.1 b2HIT	159
11.30.2.2 b2C	159
11.30.2.3 b2Shapiro	159
11.30.2.4 add_root	159
11.30.2.5 expand_Shapiro	160
11.30.2.6 expand_Full	160
11.30.2.7 unexpand_Full	160
11.30.2.8 unweight	160
11.30.2.9 unexpand_aligned_F	160
11.30.2.10 parse_structure	161
11.31/home/mescal/ronny/WORK/ViennaRNA/H/stringdist.h File Reference	161
11.31.1 Detailed Description	161
11.31.2 Function Documentation	161
11.31.2.1 Make_swString	162
11.31.2.2 string_edit_distance	162
11.32/home/mescal/ronny/WORK/ViennaRNA/H/subopt.h File Reference	162
11.32.1 Detailed Description	163
11.33/home/mescal/ronny/WORK/ViennaRNA/H/treedist.h File Reference	163
11.33.1 Detailed Description	164
11.33.2 Function Documentation	164
11.33.2.1 make_tree	164
11.33.2.2 tree_edit_distance	164
11.33.2.3 free_tree	164
11.34/home/mescal/ronny/WORK/ViennaRNA/H/Utils.h File Reference	164
11.34.1 Detailed Description	167
11.34.2 Macro Definition Documentation	167

11.34.2.1	VRNA_INPUT_ERROR	167
11.34.2.2	VRNA_INPUT_QUIT	167
11.34.2.3	VRNA_INPUT_MISC	167
11.34.2.4	VRNA_INPUT_FASTA_HEADER	167
11.34.2.5	VRNA_INPUT_SEQUENCE	167
11.34.2.6	VRNA_INPUT_CONSTRAINT	167
11.34.2.7	VRNA_INPUT_NO_TRUNCATION	168
11.34.2.8	VRNA_INPUT_NO_REST	168
11.34.2.9	VRNA_INPUT_NO_SPAN	168
11.34.2.10	VRNA_INPUT_NOSKIP_BLANK_LINES	168
11.34.2.11	VRNA_INPUT_BLANK_LINE	168
11.34.2.12	VRNA_INPUT_NOSKIP_COMMENTS	168
11.34.2.13	VRNA_INPUT_COMMENT	168
11.34.2.14	VRNA_CONSTRAINT_PIPE	168
11.34.2.15	VRNA_CONSTRAINT_DOT	168
11.34.2.16	VRNA_CONSTRAINT_X	168
11.34.2.17	VRNA_CONSTRAINT_ANG_BRACK	168
11.34.2.18	VRNA_CONSTRAINT_RND_BRACK	168
11.34.2.19	VRNA_CONSTRAINT_MULTILINE	169
11.34.2.20	VRNA_CONSTRAINT_NO_HEADER	169
11.34.2.21	VRNA_CONSTRAINT_ALL	169
11.34.2.22	VRNA_CONSTRAINT_G	169
11.34.2.23	VRNA_OPTION_MULTILINE	169
11.34.2.24	MIN2	169
11.34.2.25	MAX2	169
11.34.2.26	MIN3	169
11.34.2.27	MAX3	169
11.34.2.28	XSTR	169
11.34.2.29	STR	169
11.34.2.30	FILENAME_MAX_LENGTH	170
11.34.2.31	FILENAME_ID_LENGTH	170
11.34.3	Function Documentation	170
11.34.3.1	space	170
11.34.3.2	xrealloc	170
11.34.3.3	nrerror	170
11.34.3.4	warn_user	171
11.34.3.5	urn	171
11.34.3.6	int_urn	171
11.34.3.7	time_stamp	171
11.34.3.8	random_string	171

11.34.3.9 hamming	172
11.34.3.10 hamming_bound	172
11.34.3.11 get_line	172
11.34.3.12 get_input_line	172
11.34.3.13 read_record	173
11.34.3.14 pack_structure	174
11.34.3.15 unpack_structure	174
11.34.3.16 make_pair_table	174
11.34.3.17 copy_pair_table	175
11.34.3.18 alimake_pair_table	175
11.34.3.19 make_pair_table_snoop	175
11.34.3.20 make_loop_index_pt	175
11.34.3.21 print_tty_input_seq	175
11.34.3.22 print_tty_input_seq_str	175
11.34.3.23 print_tty_constraint	176
11.34.3.24 str_DNA2RNA	176
11.34.3.25 str_uppercase	176
11.34.3.26 get_iindx	176
11.34.3.27 get_indx	177
11.34.3.28 constrain_ptypes	177
11.34.4 Variable Documentation	178
11.34.4.1 xsubi	178
11.35/home/mescal/ronny/WORK/ViennaRNA/lib/1.8.4_epars.h File Reference	178
11.35.1 Detailed Description	178
11.36/home/mescal/ronny/WORK/ViennaRNA/lib/1.8.4_intloops.h File Reference	178
11.36.1 Detailed Description	178

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 ([7],[5]) 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[3] 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[11], which, does not retain the full information of the secondary structure. He represents the different structure elements by single matching brackets and labels them as 'H' (hairpin loop), 'I' (interior loop), 'B' (bulge), 'M' (multi-loop), and 'S' (stack). We extend his alphabet by an extra letter for external elements 'E'. Again these identifiers may be followed by a weight corresponding to the number of unpaired bases or base pairs in the structure element. All tree representations (except for the dot-bracket form) can be encapsulated into a virtual root (labeled 'R'), see the example below.

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

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

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

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

```
a) (UU) (P (P (P (P (UU) (UU) (P (P (P (UU) (UU) (UU) P) P) P) (UU) (UU) (P (P (UU) (U...
b) (UU) (P2 (P2 (U2U2) (P2 (U3U3) P3) (U2U2) (P2 (U2U2) P2) P2) (UU) P2) (UU)
c) (B (M (HH) (HH) M) B)
```

```
(S (B (S (M (S (HH) S) (S (HH) S) M) S) B) S)
(E (S (B (S (M (S (HH) S) (S (HH) S) M) S) B) S) E)
d) (R (E2 (S2 (B1 (S2 (M4 (S3 (H3) S3) ((H2) S2) M4) S2) B1) S2) E2) R)
```

Aligned structures additionally contain the gap character '_'.

Parsing and Coarse Graining of Structures

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

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

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

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'.

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.

```
#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 *pfl, *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](#) (int length, double *dist)

This function is deprecated and should not be used anymore as it is not threadsafe!

Global [energy_of_circ_struct](#) (const char *string, const char *structure)

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

Global [energy_of_struct](#) (const char *string, const char *structure)

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

Global [energy_of_struct_pt](#) (const char *string, short *ptable, short *s, short *s1)

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

Global [expHairpinEnergy](#) (int u, int type, short si1, short sj1, const char *string)

Use [exp_E_Hairpin\(\)](#) from [loop_energies.h](#) instead

Global [expLoopEnergy](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)

Use [exp_E_IntLoop\(\)](#) from [loop_energies.h](#) instead

Global [get_plist](#) (struct plist *pl, int length, double cut_off)

{ This function is deprecated and will be removed soon!} use [assign_plist_from_pr\(\)](#) instead!

Global [HairpinE](#) (int size, int type, int si1, int sj1, const char *string)

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

Global [iindx](#)

Do not use this variable anymore!

Global [init_co_pf_fold](#) (int length)

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

Global [init_pf_fold](#) (int length)

This function is obsolete and will be removed soon!

Global [initialize_cofold](#) (int length)

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

Global [initialize_fold](#) (int length)

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

Global [LoopEnergy](#) (int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1)

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

Global [Make_bp_profile](#) (int length)

This function is deprecated and will be removed soon! See [Make_bp_profile_bppm\(\)](#) for a replacement

Global [mean_bp_dist](#) (int length)

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](#) (char *string, char *file)

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	17
Calculating Minimum Free Energy (MFE) Structures	20
MFE Structures of two hybridized Sequences	40
MFE Consensus Structures for Sequence Alignment(s)	52
Local MFE structure Prediction and Z-scores	58
Calculating MFE representatives of a Distance Based Partitioning	79
Calculating Partition Functions and Pair Probabilities	23
Compute the structure with maximum expected accuracy (MEA)	30
Compute the centroid structure	31
Partition Function for two hybridized Sequences	42
Partition Function for two hybridized Sequences as a stepwise Process	46
Partition Function and Base Pair Probabilities for Sequence Alignment(s)	54
Partition functions for locally stable secondary structures	59
Calculate Partition Functions of a Distance Based Partitioning	82
Enumerating Suboptimal Structures	33
Suboptimal structures according to Zuker et al. 1989	34
Suboptimal structures within an energy band around the MFE	35
Stochastic backtracking in the Ensemble	37
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	56
Stochastic Backtracking of Structures from Distance Based Partitioning	85
Calculate Secondary Structures of two RNAs upon Dimerization	39
MFE Structures of two hybridized Sequences	40
Partition Function for two hybridized Sequences	42
Partition Function for two hybridized Sequences as a stepwise Process	46
Predicting Consensus Structures from Alignment(s)	48
MFE Consensus Structures for Sequence Alignment(s)	52
Partition Function and Base Pair Probabilities for Sequence Alignment(s)	54
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	56
Local MFE consensus structures for Sequence Alignments	61
Predicting Locally stable structures of large sequences	57
Local MFE structure Prediction and Z-scores	58
Partition functions for locally stable secondary structures	59
Local MFE consensus structures for Sequence Alignments	61
Change and Precalculate Energy Parameter Sets and Boltzmann Factors	62
Reading/Writing energy parameter sets from/to File	65
Converting energy parameter files	67
Energy evaluation	71

Searching Sequences for Predefined Structures	75
Classified Dynamic Programming	77
Distance based partitioning of the Secondary Structure Space	78
Calculating MFE representatives of a Distance Based Partitioning	79
Calculate Partition Functions of a Distance Based Partitioning	82
Stochastic Backtracking of Structures from Distance Based Partitioning	85
Compute the Density of States	87
Parsing and Comparing - Functions to Manipulate Structures	88

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

bondT	Base pair	89
bondTEn	Base pair with associated energy	89
cofoldF	89
ConcEnt	90
constrain	Constraints for cofolding	90
COORDINATE	This is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type COORDINATE	90
cpair	This datastructure is used as input parameter in functions of PS_dot.c	90
duplexT	91
dupVar	91
folden	91
interact	91
intermediate_t	92
INTERVAL	Sequence interval stack element used in subopt.c	92
LIST	93
LST_BUCKET	93
model_detailsT	The data structure that contains the complete model details used throughout the calculations	93
move_t	94
PAIR	Base pair data structure used in subopt.c	94
pair_info	A base pair info structure	95
pairpro	96
paramT	The datastructure that contains temperature scaled energy parameters	96
path_t	97
pf_paramT	The datastructure that contains temperature scaled Boltzmann weights of the energy parameters	97
plist	This datastructure is used as input parameter in functions of PS_dot.h and others	98
Postorder_list	98

pu_contrib		
	Contributions to p_u	98
pu_out		
	Collection of all free_energy of beeing unpaired values for output	99
sect		
	Stack of partial structures for backtracking	99
snoopT		99
SOLUTION		
	Solution element from subopt.c	100
struct_en		100
svm_model		100
swString		100
Tree		101
TwoDfold_solution		
	Solution element returned from TwoDfoldList	101
TwoDfold_vars		
	Variables compound for 2Dfold MFE folding	102
TwoDpfold_solution		
	Solution element returned from TwoDpfoldList	103
TwoDpfold_vars		
	Variables compound for 2Dfold partition function folding	103

Chapter 8

File Index

8.1 File List

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

mainpage.h	??
/home/mescalini/ronny/WORK/ViennaRNA/H/2Dfold.h	107
/home/mescalini/ronny/WORK/ViennaRNA/H/2Dpfold.h	108
/home/mescalini/ronny/WORK/ViennaRNA/H/ali_plex.h	??
/home/mescalini/ronny/WORK/ViennaRNA/H/alifold.h	
Compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments	108
/home/mescalini/ronny/WORK/ViennaRNA/H/aln_util.h	??
/home/mescalini/ronny/WORK/ViennaRNA/H/cofold.h	
MFE version of cofolding routines	110
/home/mescalini/ronny/WORK/ViennaRNA/H/convert_epars.h	
Functions and definitions for energy parameter file format conversion	112
/home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h	
All datastructures and typedefs shared among the Vienna RNA Package can be found here	113
/home/mescalini/ronny/WORK/ViennaRNA/H/dist_vars.h	
Global variables for Distance-Package	115
/home/mescalini/ronny/WORK/ViennaRNA/H/duplex.h	
Duplex folding function declarations..	116
/home/mescalini/ronny/WORK/ViennaRNA/H/edit_cost.h	
Global variables for Edit Costs included by treedist.c and stringdist.c	116
/home/mescalini/ronny/WORK/ViennaRNA/H/energy_const.h	117
/home/mescalini/ronny/WORK/ViennaRNA/H/energy_par.h	??
/home/mescalini/ronny/WORK/ViennaRNA/H/findpath.h	
Compute direct refolding paths between two secondary structures	118
/home/mescalini/ronny/WORK/ViennaRNA/H/fold.h	
MFE calculations and energy evaluations for single RNA sequences	120
/home/mescalini/ronny/WORK/ViennaRNA/H/fold_vars.h	
Here all all declarations of the global variables used throughout RNAlib	126
/home/mescalini/ronny/WORK/ViennaRNA/H/gquad.h	
Various functions related to G-quadruplex computations	131
/home/mescalini/ronny/WORK/ViennaRNA/H/inverse.h	
Inverse folding routines	133
/home/mescalini/ronny/WORK/ViennaRNA/H/Lfold.h	
Predicting local MFE structures of large sequences	133
/home/mescalini/ronny/WORK/ViennaRNA/H/loop_energies.h	
Energy evaluation for MFE and partition function calculations	134
/home/mescalini/ronny/WORK/ViennaRNA/H/LPfold.h	
Function declarations of partition function variants of the Lfold algorithm	139

/home/mescal/ronny/WORK/ViennaRNA/H/MEA.h	
Computes a MEA (maximum expected accuracy) structure	140
/home/mescal/ronny/WORK/ViennaRNA/H/mm.h	
Several Maximum Matching implementations	141
/home/mescal/ronny/WORK/ViennaRNA/H/move_set.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/naview.h	141
/home/mescal/ronny/WORK/ViennaRNA/H/pair_mat.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/params.h	142
/home/mescal/ronny/WORK/ViennaRNA/H/part_func.h	
Partition function of single RNA sequences	143
/home/mescal/ronny/WORK/ViennaRNA/H/part_func_co.h	
Partition function for two RNA sequences	145
/home/mescal/ronny/WORK/ViennaRNA/H/part_func_up.h	
Partition Function Cofolding as stepwise process	147
/home/mescal/ronny/WORK/ViennaRNA/H/PKplex.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/plex.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/plot_layouts.h	
Secondary structure plot layout algorithms	148
/home/mescal/ronny/WORK/ViennaRNA/H/ProfileAln.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/profiledist.h	152
/home/mescal/ronny/WORK/ViennaRNA/H/PS_dot.h	
Various functions for plotting RNA secondary structures, dot-plots and other visualizations . . .	153
/home/mescal/ronny/WORK/ViennaRNA/H/read_epars.h	157
/home/mescal/ronny/WORK/ViennaRNA/H/ribo.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/RNAstruct.h	
Parsing and Coarse Graining of Structures	157
/home/mescal/ronny/WORK/ViennaRNA/H/snofold.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/snoop.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/stringdist.h	
Functions for String Alignment	161
/home/mescal/ronny/WORK/ViennaRNA/H/subopt.h	
RNAsubopt and density of states declarations	162
/home/mescal/ronny/WORK/ViennaRNA/H/svm_utils.h	??
/home/mescal/ronny/WORK/ViennaRNA/H/treedist.h	
Functions for Tree Edit Distances	163
/home/mescal/ronny/WORK/ViennaRNA/H/utils.h	
Various utility- and helper-functions used throughout the Vienna RNA package	164
/home/mescal/ronny/WORK/ViennaRNA/lib/1.8.4_epars.h	
Free energy parameters for parameter file conversion	178
/home/mescal/ronny/WORK/ViennaRNA/lib/1.8.4_intloops.h	
Free energy parameters for interior loop contributions needed by the parameter file conversion functions	178
/home/mescal/ronny/WORK/ViennaRNA/lib/intl11.h	??
/home/mescal/ronny/WORK/ViennaRNA/lib/intl11dH.h	??
/home/mescal/ronny/WORK/ViennaRNA/lib/intl21.h	??
/home/mescal/ronny/WORK/ViennaRNA/lib/intl21dH.h	??
/home/mescal/ronny/WORK/ViennaRNA/lib/intl22.h	??
/home/mescal/ronny/WORK/ViennaRNA/lib/intl22dH.h	??
/home/mescal/ronny/WORK/ViennaRNA/lib/list.h	??

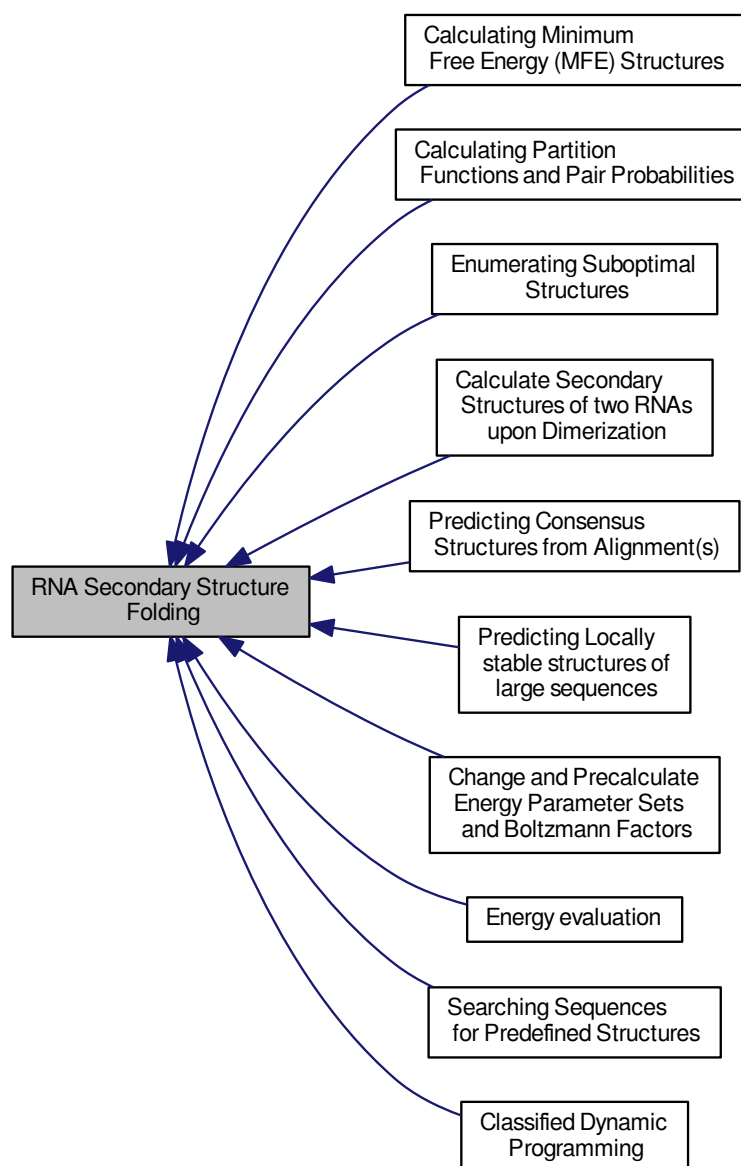
Chapter 9

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](#)

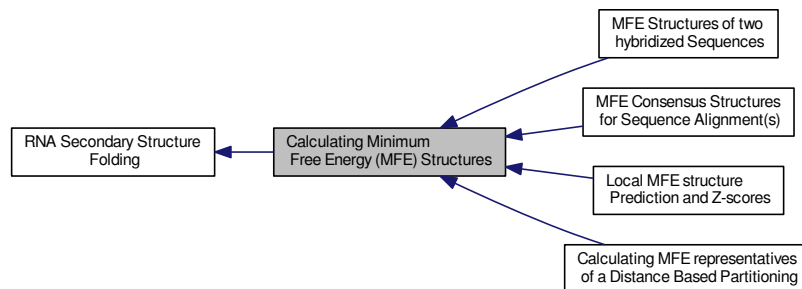
9.1.1 Detailed Description

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[13]. 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[6] 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 Σ that is understood by the RNAlib

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

The second parameter, *structure*, must always point to an allocated block of memory with a size of at least `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

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details. (NULL may be passed, see OpenMP notes above)
<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
<i>is_circular</i>	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

<i>sequence</i>	RNA sequence
<i>structure</i>	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

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

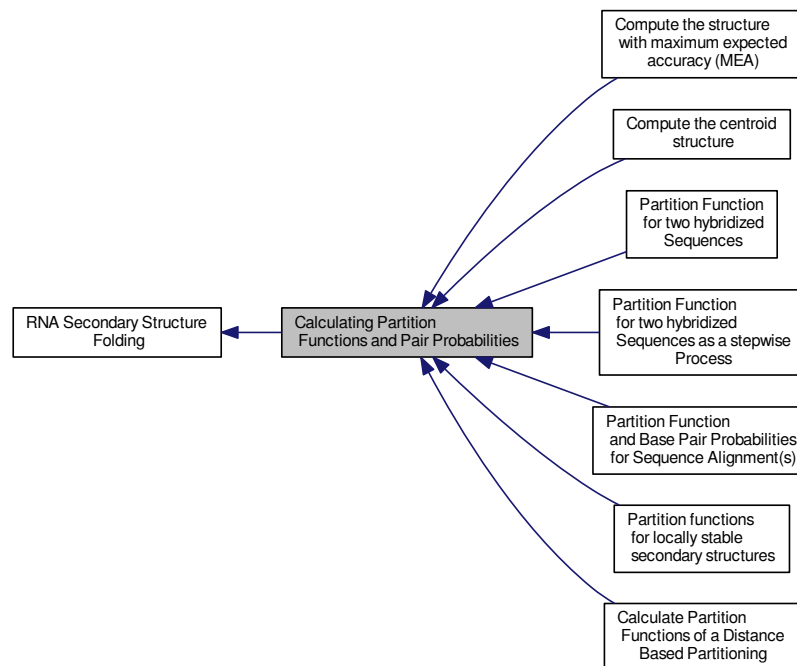
Returns

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.

Files

- file [part_func.h](#)
Partition function of single RNA sequences.

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[10].

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

<code>in</code>	<code>sequence</code>	The RNA sequence input
<code>in, out</code>	<code>structure</code>	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)
<code>in</code>	<code>parameters</code>	Data structure containing the precalculated Boltzmann factors
<code>in</code>	<code>calculate_bppm</code>	Switch to Base pair probability calculations on/off (0==off)
<code>in</code>	<code>is_constrained</code>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
<code>in</code>	<code>is_circular</code>	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

<i>sequence</i>	The RNA sequence input
<i>structure</i>	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

<i>in</i>	<i>sequence</i>	The RNA sequence input
<i>in, out</i>	<i>structure</i>	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

out	<i>pl</i>	A pointer to the plist that is to be created
in	<i>probs</i>	The probability matrix used for creting the plist
in	<i>length</i>	The length of the RNA sequence
in	<i>cutoff</i>	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 relavant computation arrays used in partition function computation.

Precondition

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

See Also

[pf_fold_par\(\)](#), [pf_fold\(\)](#), [pf_circ_fold\(\)](#)

Parameters

out	<i>S_p</i>	A pointer to the 'S' array (integer representation of nucleotides)
out	<i>S1_p</i>	A pointer to the 'S1' array (2nd integer representation of nucleotides)
out	<i>ptype_p</i>	A pointer to the pair type matrix
out	<i>qb_p</i>	A pointer to the Q^B matrix
out	<i>qm_p</i>	A pointer to the Q^M matrix
out	<i>q1k_p</i>	A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1,k)$)
out	<i>qln_p</i>	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

<i>length</i>	
---------------	--

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 $p_{i,j}$ as

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

Note

This function is threadsafe

Parameters

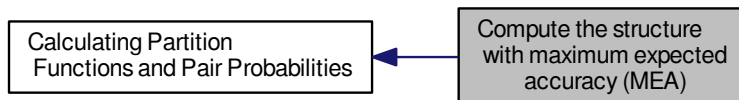
<i>length</i>	The length of the sequence
<i>pr</i>	The matrix containing the base pair probabilities

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 Detailed Description

9.5.2 Function Documentation

9.5.2.1 `char* get_centroid_struct_pl (int length, double * dist, plist * pl)`

Get the centroid structure of the ensemble.

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

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

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

Thus, the centroid is simply the structure containing all pairs with $p_{ij} > 0.5$ The distance of the centroid to the ensemble is written to the memory addressed by *dist*.

Parameters

in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>pl</i>	A pair list containing base pair probability information about the ensemble

Returns

The centroid structure of the ensemble in dot-bracket notation

9.5.2.2 `char* get_centroid_struct_pr (int length, double * dist, double * pr)`

Get the centroid structure of the ensemble.

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

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

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

Thus, the centroid is simply the structure containing all pairs with $p_{i,j} > 0.5$. The distance of the centroid to the ensemble is written to the memory addressed by *dist*.

Parameters

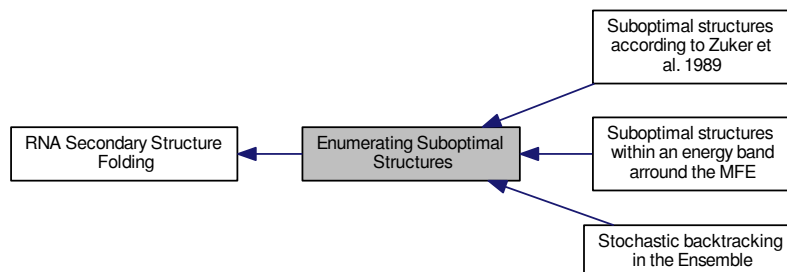
in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>pr</i>	A upper triangular matrix containing base pair probabilities (access via <code>iindx</code> get_iindx())

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](#)

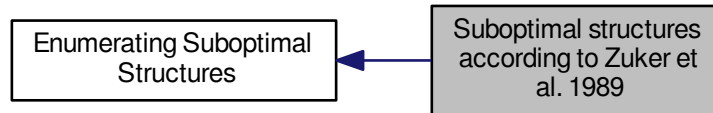
Files

- file [subopt.h](#)
RNAsubopt and density of states declarations.

9.6.1 Detailed Description

9.7 Suboptimal structures according to Zuker et al. 1989

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



Functions

- **SOLUTION** * `zukersubopt` (const char *string)
Compute Zuker type suboptimal structures.
- **SOLUTION** * `zukersubopt_par` (const char *string, `paramT` *parameters)
Compute Zuker type suboptimal structures.

9.7.1 Detailed Description

9.7.2 Function Documentation

9.7.2.1 **SOLUTION*** `zukersubopt` (const char * *string*)

Compute Zuker type suboptimal structures.

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

Parameters

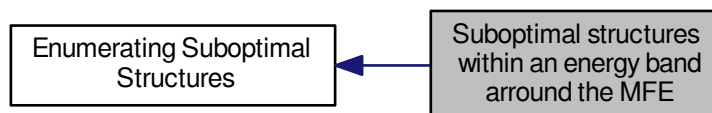
<i>string</i>	RNA sequence
---------------	--------------

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 Detailed Description

9.8.2 Function Documentation

9.8.2.1 **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

<i>seq</i>	
<i>structure</i>	
<i>delta</i>	
<i>fp</i>	

Returns

9.8.2.2 **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

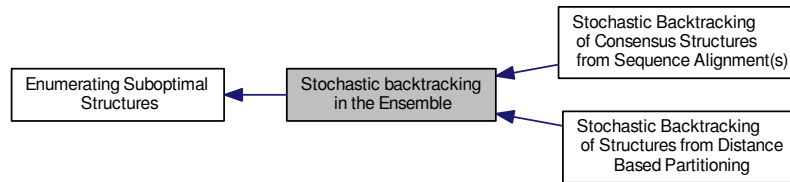
Parameters

<i>seq</i>	
<i>sequence</i>	
<i>delta</i>	
<i>fp</i>	

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 Detailed Description

9.9.2 Function Documentation

9.9.2.1 `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

<code>sequence</code>	The RNA sequence
-----------------------	------------------

Returns

A sampled secondary structure in dot-bracket notation

9.9.2.2 char* pbacktrack_circ (char * *sequence*)

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

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

Precondition

[st_back](#) has to be set to 1 before calling [pf_fold\(\)](#) or [pf_fold_par\(\)](#)

[pf_fold_par\(\)](#) or [pf_circ_fold\(\)](#) have to be called first to fill the partition function matrices

Parameters

<i>sequence</i>	The RNA sequence
-----------------	------------------

Returns

A sampled secondary structure in dot-bracket notation

9.9.3 Variable Documentation**9.9.3.1 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

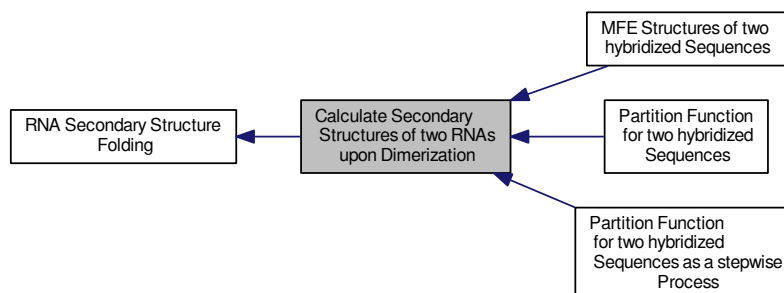
See Also

[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

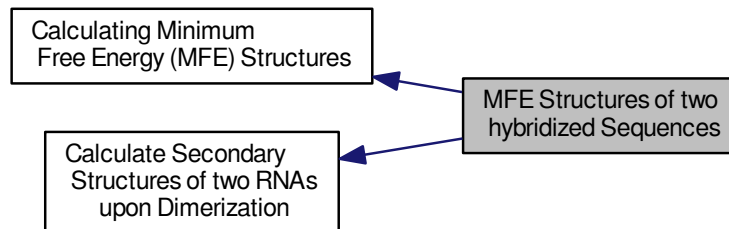
Predict structures formed by two molecules upon hybridization. The function of an RNA molecule often depends on its interaction with other RNAs. The following routines therefore allow to predict structures formed by two RNA molecules upon hybridization.

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 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.

9.11.1 Detailed Description

9.11.2 Function Documentation

9.11.2.1 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

<i>sequence</i>	The two sequences concatenated
<i>structure</i>	Will hold the barcket dot structure of the dimer molecule

Returns

minimum free energy of the structure

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

Export the arrays of partition function cofold (with gquadraplex support)

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

Parameters

<i>f5_p</i>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<i>c_p</i>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<i>fML_p</i>	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
<i>fM1_p</i>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<i>fc_p</i>	A pointer to the 'fc' array, i.e. array ...
<i>ggg_p</i>	A pointer to the 'ggg' array, i.e. array containing best free energy of a gquadraplex delimited by [i..j]
<i>indx_p</i>	A pointer to the indexing array used for accessing the energy matrices
<i>ptype_p</i>	A pointer to the ptype array containing the base pair types for each possibility (i..j)

9.11.2.3 `void export_cofold_arrays (int ** f5_p, int ** c_p, int ** fML_p, int ** fM1_p, int ** fc_p, int ** indx_p, char ** ptype_p)`

Export the arrays of partition function cofold.

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

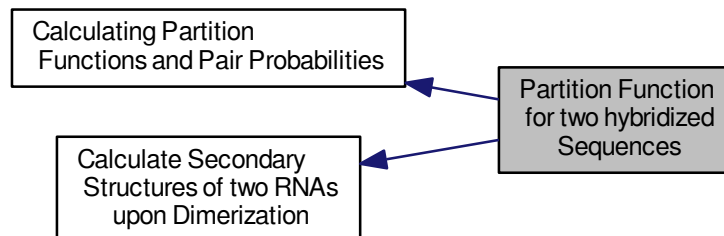
Parameters

<i>f5_p</i>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<i>c_p</i>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<i>fML_p</i>	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
<i>fM1_p</i>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<i>fc_p</i>	A pointer to the 'fc' array, i.e. array ...
<i>indx_p</i>	A pointer to the indexing array used for accessing the energy matrices
<i>ptype_p</i>	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[2] 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

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	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

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Pointer to the structure constraint
<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
<i>calculate_bppm</i>	Switch to turn Base pair probability calculations on/off (0==off)
<i>is_constrained</i>	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

<i>length</i>	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. Its 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

<i>length</i>	Length of the current RNA sequence
<i>parameters</i>	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

<i>FAB</i>	free energy of dimer AB
<i>FEA</i>	free energy of monomer A
<i>FEB</i>	free energy of monomer B
<i>prAB</i>	pair probabilities for dimer
<i>prA</i>	pair probabilities monomer
<i>prB</i>	pair probabilities monomer
<i>Alength</i>	Length of molecule A

9.12.2.7 **ConcEnt* get_concentrations (double *FEAB*, double *FEAA*, double *FEBC*, 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

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

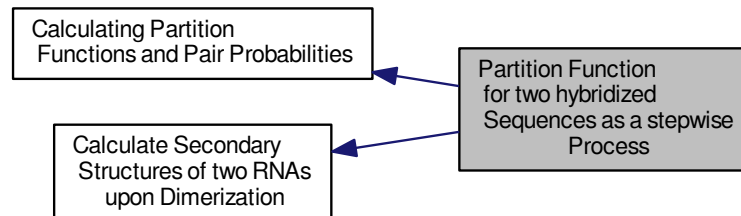
Returns

[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 to length(sequence)][j = 0 to u-1] containing all possible contributions to the probabilities of unpaired regions of maximum length u. Each array in [pu_contrib](#) contains one of the contributions to the total probability of being unpaired: The probability of being unpaired within an exterior loop is in array [pu_contrib](#)->E, the probability of being unpaired within a hairpin loop is in array [pu_contrib](#)->H, the probability of being unpaired within an interior loop is in array [pu_contrib](#)->I and probability of being

unpaired within a multi-loop is in array `pu_contrib->M`. The total probability of being unpaired is the sum of the four arrays of `pu_contrib`.

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

Parameters

<i>sequence</i>	
<i>max_w</i>	

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 constraints 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_{ijkl} 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_{ijkl_wo} is G_{ijkl} without the probability of being unpaired.

Use `free_interact()` to free the returned structure, all other stuff is freed inside `pf_interact()`.

Parameters

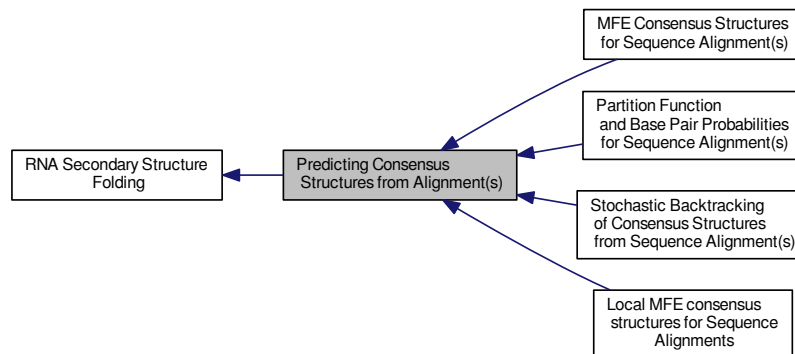
<i>s1</i>	
<i>s2</i>	
<i>p_c</i>	
<i>p_c2</i>	
<i>max_w</i>	
<i>cstruc</i>	
<i>incr3</i>	
<i>incr5</i>	

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](#)

Files

- file [alifold.h](#)
compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA 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_al_sequence](#) (const char *sequence, short *S, short *s5, short *s3, char *ss, unsigned short *as, int circ)
Get arrays with encoded sequence of the alignment.
- void [alloc_sequence_arrays](#) (const char **sequences, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss, int circ)
Allocate memory for sequence array used to deal with aligned sequences.
- void [free_sequence_arrays](#) (unsigned int n_seq, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss)
Free the memory of the sequence arrays used to deal with aligned sequences.

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

Get pointers to (almost) all 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[\[4\]](#) and [\[1\]](#).

9.14.2 Function Documentation

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

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

Parameters

<i>Alseq</i>	
<i>n_seq</i>	The number of sequences in the alignment
<i>length</i>	The length of the alignment
<i>mini</i>	

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

<i>sequences</i>	The NULL terminated array of sequences
<i>structure</i>	The consensus structure
<i>n_seq</i>	The number of sequences in the alignment
<i>energy</i>	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

<i>sequence</i>	The gapped sequence from the alignment
<i>S</i>	pointer to an array that holds encoded sequence
<i>s5</i>	pointer to an array that holds the next base 5' of alignment position i
<i>s3</i>	pointer to an array that holds the next base 3' of alignment position i
<i>ss</i>	
<i>as</i>	
<i>circ</i>	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

<i>sequences</i>	The aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position
<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	A pointer to the array that contains the ungapped sequence
<i>circ</i>	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

<i>n_seq</i>	The number of aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position
<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	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,
double ** qb_p, double ** qm_p, double ** q1k_p, double ** 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

<code>S_p</code>	A pointer to the 'S' array (integer representation of nucleotides)
<code>S5_p</code>	A pointer to the 'S5' array
<code>S3_p</code>	A pointer to the 'S3' array
<code>a2s_p</code>	A pointer to the pair type matrix
<code>Ss_p</code>	A pointer to the 'Ss' array
<code>qb_p</code>	A pointer to the Q^B matrix
<code>qm_p</code>	A pointer to the Q^M matrix
<code>q1k_p</code>	A pointer to the 5' slice of the Q matrix ($q1k(k) = Q(1,k)$)
<code>qln_p</code>	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.

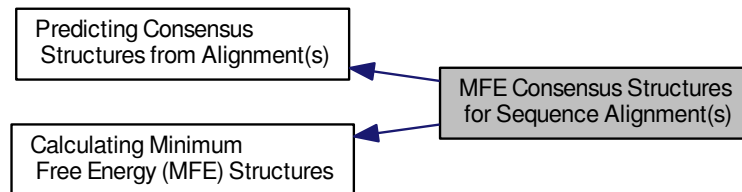
9.14.3.2 double nc_fact

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 [circaifold](#) (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 Detailed Description

9.15.2 Function Documentation

9.15.2.1 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

<i>strings</i>	A pointer to a NULL terminated array of character arrays
<i>structure</i>	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

9.15.2.2 float circaifold (const char ** strings, char * structure)

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

Parameters

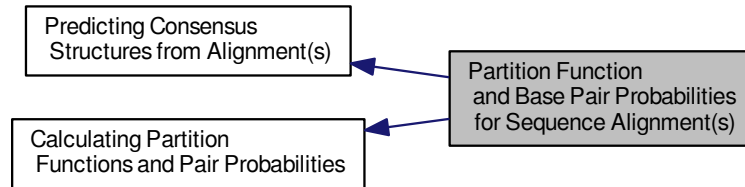
<i>strings</i>	A pointer to a NULL terminated array of character arrays
<i>structure</i>	A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

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)
- double * [export_ali_bppm](#) (void)

Get a pointer to the base pair probability array.

9.16.1 Detailed Description

9.16.2 Function Documentation

9.16.2.1 float [alipf_fold_par](#) (const char ** sequences, char * structure, [plist](#) ** pl, [pf_paramT](#) * parameters, int calculate_bppm, int is_constrained, int is_circular)

Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	
<i>parameters</i>	
<i>calculate_bppm</i>	
<i>is_constrained</i>	
<i>is_circular</i>	

Returns

9.16.2.2 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

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

Returns

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

Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

Returns

9.16.2.4 double* export_ali_bppm (void)

Get a pointer to the base pair probability array.

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

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

See Also

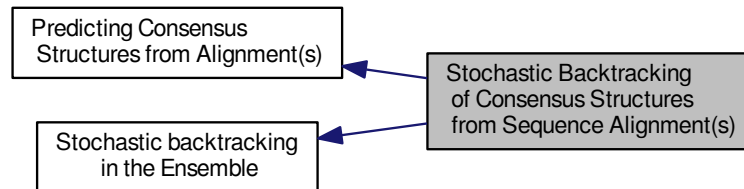
[get_iindx\(\)](#)

Returns

A pointer to the base pair probability array

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 Detailed Description

9.17.2 Function Documentation

9.17.2.1 char* alipbacktrack (double * prob)

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

.

Parameters

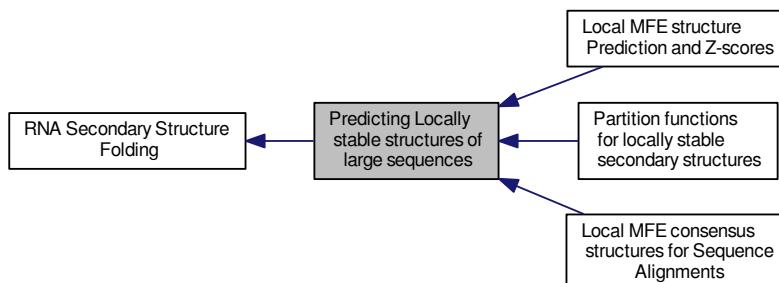
<i>prob</i>	to be described (berni)
-------------	-------------------------

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](#)

Files

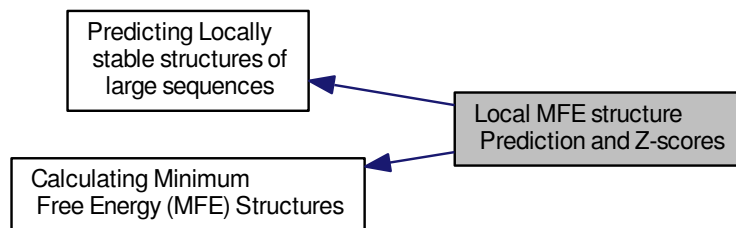
- file [Lfold.h](#)
Predicting local MFE structures of large sequences.

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 Detailed Description

9.19.2 Function Documentation

9.19.2.1 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

<i>string</i>	
<i>structure</i>	
<i>maxdist</i>	

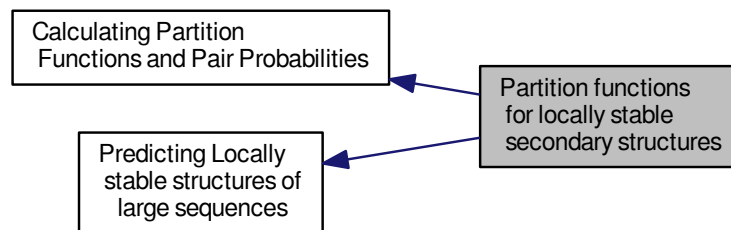
9.19.2.2 float Lfoldz (const char * *string*, char * *structure*, int *maxdist*, int *zsc*, double *min_z*)

Parameters

<i>string</i>	
<i>structure</i>	
<i>maxdist</i>	
<i>zsc</i>	
<i>min_z</i>	

9.20 Partition functions for locally stable secondary structures

Collaboration diagram for Partition functions for locally stable secondary structures:



Files

- file [LPfold.h](#)

Function declarations of partition function variants of the Lfold algorithm.

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 Detailed Description

9.20.2 Function Documentation

9.20.2.1 void [update_pf_paramsLP](#) (int length)

Parameters

<i>length</i>	
---------------	--

9.20.2.2 [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

<i>sequence</i>	RNA sequence
<i>winSize</i>	size of the window
<i>pairSize</i>	maximum size of base pair
<i>cutoffb</i>	cutoffb for base pairs
<i>pU</i>	array holding all unpaired probabilities
<i>dpp2</i>	array of dependent pair probabilities
<i>pUfp</i>	file pointer for pU
<i>spup</i>	file pointer for pair probabilities

Returns

list of pair probabilities

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

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

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

Parameters

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

9.20.2.4 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.

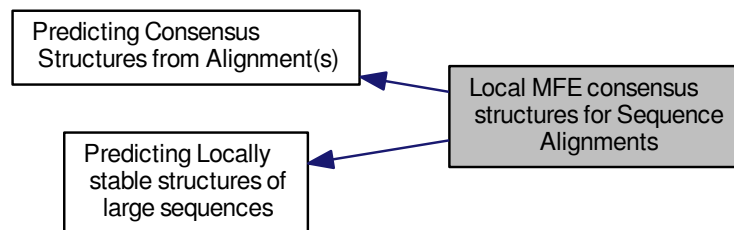
Can write either the unpaired probabilities (accessibilities) `pU` or the opening energies `-log(pU)kT` into a file

Parameters

<i>pU</i>	pair probabilities
<i>length</i>	length of RNA sequence
<i>ulength</i>	maximum length of unpaired stretch
<i>fp</i>	file pointer of destination file
<i>energies</i>	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 Detailed Description

9.21.2 Function Documentation

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

Parameters

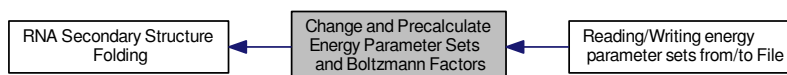
<i>strings</i>	
<i>structure</i>	
<i>maxdist</i>	

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)
- PUBLIC [pf_paramT * get_boltzmann_factors_alif](#) (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.

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

<i>temperature</i>	The temperature in degrees Celcius
<i>md</i>	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

<i>temperature</i>	The temperature in degrees Celcius used for (re-)scaling the energy contributions
<i>betaScale</i>	A scaling value that is used as a multiplication factor for the absolute temperature of the system
<i>md</i>	The model details to be used
<i>pf_scale</i>	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

<i>parameters</i>	The input data structure that shall be copied
-------------------	-----------------------------------------------

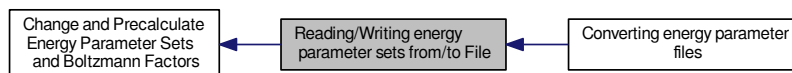
Returns

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 [9] and [12], 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

<i>fname</i>	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

<i>fname</i>	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.

Macros

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

Functions

- void [convert_parameter_file](#) (const char *iname, const char *oname, unsigned int options)

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 Macro Definition 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)
```

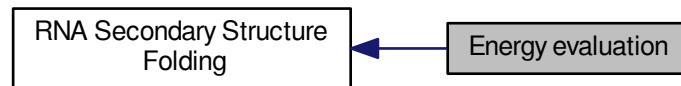
Parameters

<i>iname</i>	The input file name (If NULL input is read from stdin)
<i>oname</i>	The output file name (If NULL output is written to stdout)
<i>options</i>	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

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation
<i>verbosity_level</i>	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

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	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

<i>string</i>	RNA sequence
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	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

<i>string</i>	RNA sequence
<i>structure</i>	Secondary structure in dot-bracket notation
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	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 * s1, int verbosity_level)`

Calculate the free energy of an already folded RNA.

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

Note

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

See Also

[make_pair_table\(\)](#), [energy_of_struct_pt_par\(\)](#)

Parameters

<i>string</i>	RNA sequence
<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>verbosity_level</i>	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 * s1, paramT * parameters, int verbosity_level)`

Calculate the free energy of an already folded RNA.

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

See Also

[make_pair_table\(\)](#), [energy_of_struct_par\(\)](#), [get_scaled_parameters\(\)](#)

Parameters

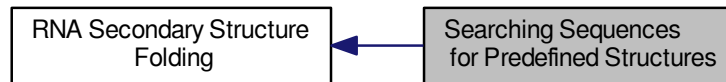
<i>string</i>	RNA sequence in uppercase letters
<i>ptable</i>	The pair table of the secondary structure
<i>s</i>	Encoded RNA sequence
<i>s1</i>	Encoded RNA sequence
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to 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

<i>start</i>	The start sequence
<i>target</i>	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

<i>start</i>	The start sequence
<i>target</i>	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`

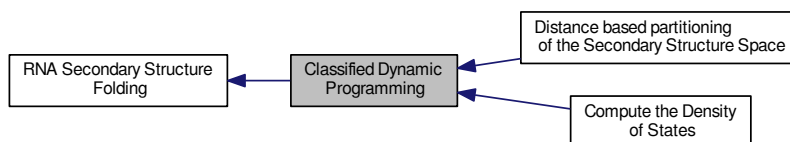
default 0: try to minimize structure distance even if no exact solution can be found

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

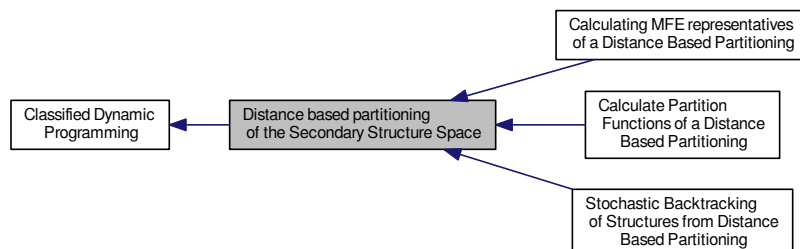
- [Distance based partitioning of the Secondary Structure Space](#)
Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.
- [Compute the Density of States](#)

9.27.1 Detailed Description

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

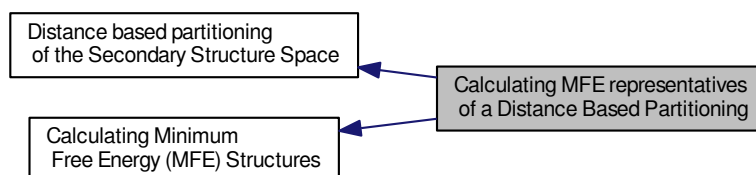
See Also

For further details have a look into[8]

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

<i>seq</i>	The RNA sequence
<i>structure1</i>	The first reference structure in dot-bracket notation
<i>structure2</i>	The second reference structure in dot-bracket notation
<i>circ</i>	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

<i>our_variables</i>	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=-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

<i>vars</i>	the datastructure containing all predefined folding attributes
<i>distance1</i>	maximum distance to reference1 (-1 means no restriction)
<i>distance2</i>	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\(\)](#)

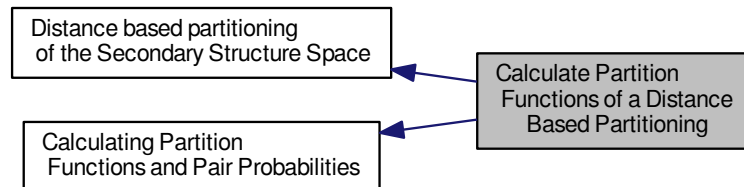
Parameters

<i>j</i>	The length in nucleotides beginning from the 5' end
<i>k</i>	distance to reference1 (may be -1)
<i>l</i>	distance to reference2
<i>vars</i>	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` (TwoDpfold_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

<i>seq</i>	the RNA sequence in uppercase format with letters from the alphabet {AUCG}
<i>structure1</i>	the first reference structure in dot-bracket notation
<i>structure2</i>	the second reference structure in dot-bracket notation
<i>circ</i>	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

<i>mfe_vars</i>	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

<i>vars</i>	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

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

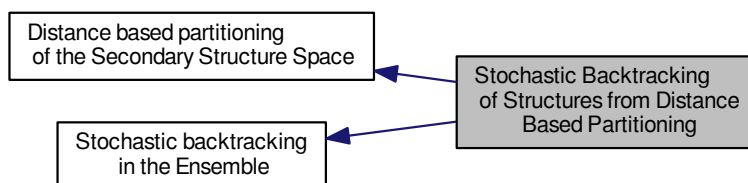
Returns

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

<code>in</code>	<code>vars</code>	the datastructure containing all necessary folding attributes and matrices
<code>in</code>	<code>d1</code>	the distance to reference1 (may be -1)
<code>in</code>	<code>d2</code>	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

<i>in</i>	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
<i>in</i>	<i>d1</i>	the distance to reference1 (may be -1)
<i>in</i>	<i>d2</i>	the distance to reference2
<i>in</i>	<i>length</i>	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 Detailed Description

9.32.2 Variable Documentation

9.32.2.1 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:

- /home/mescal/ronny/WORK/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:

- /home/mescal/ronny/WORK/ViennaRNA/H/data_structures.h

10.3 cofoldF Struct Reference

Data Fields

- double [F0AB](#)
Null model without DuplexInit.
- double [FAB](#)
all states with DuplexInit correction
- double [FcAB](#)
true hybrid states only
- double [FA](#)
monomer A

- double [FB](#)
monomer B

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

- [/home/mescal/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.4 ConcEnt Struct Reference

Data Fields

- double [A0](#)
start concentration A
- double [B0](#)
start concentration B
- double [ABc](#)
End concentration AB.

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

- [/home/mescal/ronny/WORK/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:

- [/home/mescal/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.6 COORDINATE Struct Reference

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

10.6.1 Detailed Description

this is a workaround 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:

- [/home/mescal/ronny/WORK/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:

- /home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h

10.8 duplexT Struct Reference

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

- /home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h

10.9 dupVar Struct Reference

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

- /home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h

10.10 folden Struct Reference

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

- /home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h

10.11 interact Struct Reference

Data Fields

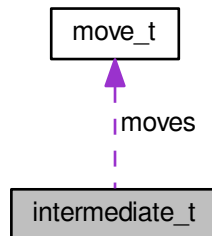
- double * [Pi](#)
probabilities of interaction
- double * [Gi](#)
free energies of interaction
- double [Gikjl](#)
full free energy for interaction between [k,i] $k < i$ in longer seq and [j,l] $j < l$ in shorter seq
- double [Gikjl_wo](#)
Gikjl without contributions for prob_unpaired.
- int [i](#)
 $k < i$ 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:

- /home/mescalini/ronny/WORK/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:

- [/home/mescal/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.13 INTERVAL Struct Reference

Sequence interval stack element used in subopt.c.

10.13.1 Detailed Description

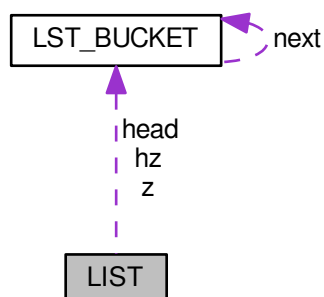
Sequence interval stack element used in subopt.c.

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

- [/home/mescal/ronny/WORK/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:

- /home/mescalini/ronny/WORK/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:

- /home/mescalini/ronny/WORK/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.
- int [canonicalBPonly](#)
remove non-canonical bp's from constraint structures

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:

- [/home/mescal/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.17 move_t Struct Reference

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

- [/home/mescal/ronny/WORK/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:

- [/home/mescal/ronny/WORK/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:

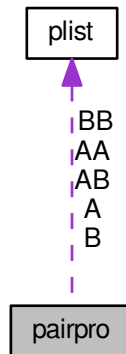
- its probability 'p'
- an entropy-like measure for its well-definedness 'ent'
- the frequency of each type of pair in 'bp[]'
 - 'bp[0]' contains the number of non-compatible sequences
 - 'bp[1]' the number of CG pairs, etc.

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

- [/home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.20 pairpro Struct Reference

Collaboration diagram for pairpro:



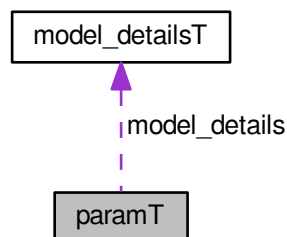
The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/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:

- /home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h

10.22 path_t Struct Reference

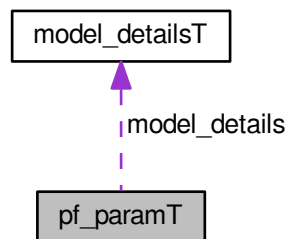
The documentation for this struct was generated from the following file:

- /home/mescalini/ronny/WORK/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:

- /home/mescalin/ronny/WORK/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:

- /home/mescalin/ronny/WORK/ViennaRNA/H/data_structures.h

10.25 Postorder_list Struct Reference

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

- /home/mescalin/ronny/WORK/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:

- [/home/mescalini/ronny/WORK/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:

- [/home/mescalini/ronny/WORK/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:

- [/home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.29 snoopT Struct Reference

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

- [/home/mescalini/ronny/WORK/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:

- [/home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h](#)

10.31 struct_en Struct Reference

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

- [/home/mescalini/ronny/WORK/ViennaRNA/H/move_set.h](#)

10.32 svm_model Struct Reference

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

- [/home/mescalini/ronny/WORK/ViennaRNA/H/svm_utils.h](#)

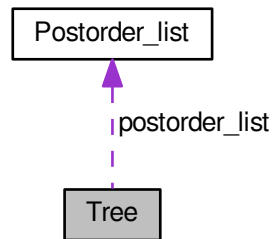
10.33 swString Struct Reference

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

- [/home/mescalini/ronny/WORK/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:

- [/home/mescalini/ronny/WORK/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 data-structure 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\(\)](#)

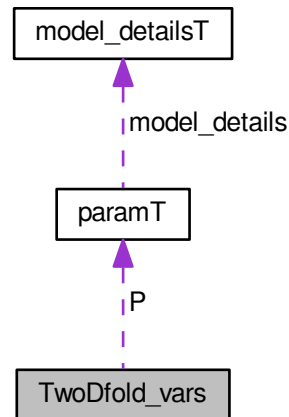
The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/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 distance 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:

- [/home/mescalini/ronny/WORK/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\(\)](#)

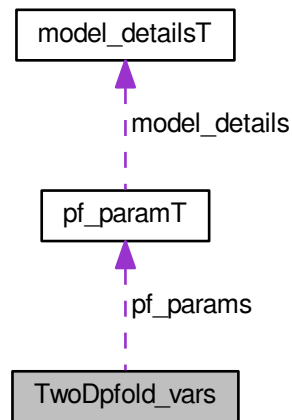
The documentation for this struct was generated from the following file:

- [/home/mescalini/ronny/WORK/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_jindx](#)
Index for moving in quadratic distance 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:

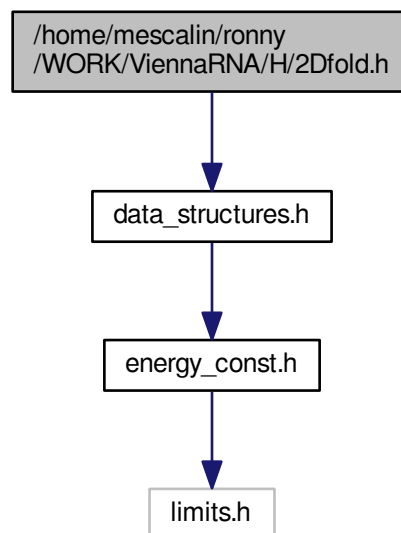
- [/home/mescalini/ronny/WORK/ViennaRNA/H/data_structures.h](#)

Chapter 11

File Documentation

11.1 /home/mescal/ronny/WORK/ViennaRNA/H/2Dfold.h File Reference

Include dependency graph for 2Dfold.h:



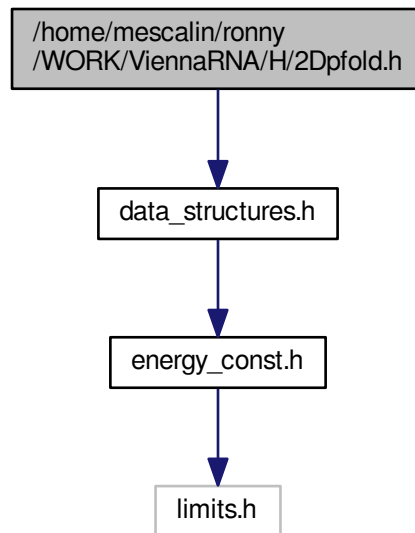
Functions

- `TwoDfold_vars * get_TwoDfold_variables` (const char *seq, const char *structure1, const char *structure2, int circ)
Get a structure of type `TwoDfold_vars` prefilled with current global settings.
- void `destroy_TwoDfold_variables` (`TwoDfold_vars` *our_variables)
Destroy a `TwoDfold_vars` datastructure without memory loss.
- `TwoDfold_solution * TwoDfoldList` (`TwoDfold_vars` *vars, int distance1, int distance2)
Compute MFE's and representative for distance partitioning.
- char * `TwoDfold_backtrack_f5` (unsigned int j, int k, int l, `TwoDfold_vars` *vars)

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

11.2 /home/mescalini/ronny/WORK/ViennaRNA/H/2Dpfold.h File Reference

Include dependency graph for 2Dpfold.h:



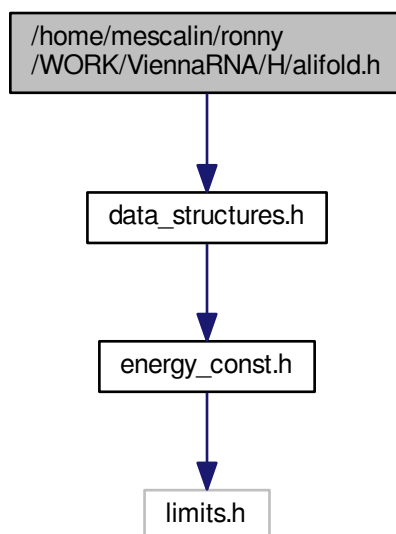
Functions

- `TwoDpfold_vars * get_TwoDpfold_variables` (const char *seq, const char *structure1, char *structure2, int circ)
Get a datastructure containing all necessary attributes and global folding switches.
- `TwoDpfold_vars * get_TwoDpfold_variables_from_MFE` (TwoDpfold_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.

11.3 /home/mescalini/ronny/WORK/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 *Alseq[], int n_seq, int length, int *mini)
Get the mean pairwise identity in steps from ?to?(ident)
- float ** [readribosum](#) (char *name)
Read a ribosum or other user-defined scoring matrix.
- float [energy_of_alistruct](#) (const char **sequences, const char *structure, int n_seq, float *energy)
Calculate the free energy of a consensus structure given a set of aligned sequences.
- void [encode_ali_sequence](#) (const char *sequence, short *S, short *s5, short *s3, char *ss, unsigned short *as, int circ)
Get arrays with encoded sequence of the alignment.
- void [alloc_sequence_arrays](#) (const char **sequences, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss, int circ)
Allocate memory for sequence array used to deal with aligned sequences.
- void [free_sequence_arrays](#) (unsigned int n_seq, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss)
Free the memory of the sequence arrays used to deal with aligned sequences.

- float [alipf_fold_par](#) (const char **sequences, char *structure, [plist](#) **pl, [pf_paramT](#) *parameters, int calculate_bppm, int is_constrained, int is_circular)
- float [alipf_fold](#) (const char **sequences, char *structure, [plist](#) **pl)

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

- float [alipf_circ_fold](#) (const char **sequences, char *structure, [plist](#) **pl)
- double * [export_ali_bppm](#) (void)

Get a pointer to the base pair probability array.

- char * [alipbacktrack](#) (double *prob)

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

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

Get pointers to (almost) all 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)

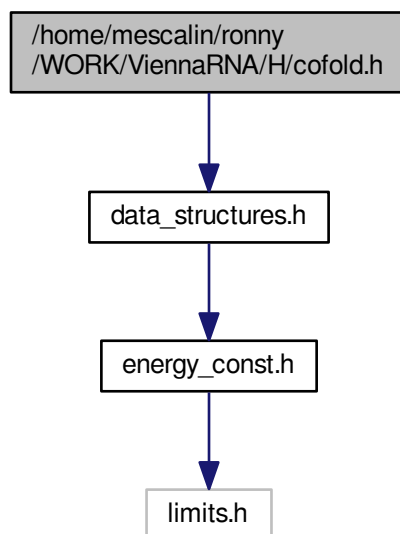
Update the energy parameters for alifold function.

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

11.4 /home/mescalini/ronny/WORK/ViennaRNA/H/cofold.h File Reference

MFE version of cofolding routines.

Include dependency graph for cofold.h:



Functions

- float [cofold](#) (const char *sequence, char *structure)
Compute the minimum free energy of two interacting RNA molecules.
- float [cofold_par](#) (const char *string, char *structure, [paramT](#) *parameters, int is_constrained)
Compute the minimum free energy of two interacting RNA molecules.
- void [free_co_arrays](#) (void)
Free memory occupied by [cofold\(\)](#)
- void [update_cofold_params](#) (void)
Recalculate parameters.
- void [export_cofold_arrays_gq](#) (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **ggg_p, int **indx_p, char **ptype_p)
Export the arrays of partition function cofold (with gquadruplex support)
- void [export_cofold_arrays](#) (int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **indx_p, char **ptype_p)
Export the arrays of partition function cofold.
- [SOLUTION](#) * [zukersubopt](#) (const char *string)
Compute Zuker type suboptimal structures.
- [SOLUTION](#) * [zukersubopt_par](#) (const char *string, [paramT](#) *parameters)
Compute Zuker type suboptimal structures.
- void [get_monomere_mfes](#) (float *e1, float *e2)
get_monomer_free_energies
- void [initialize_cofold](#) (int length)

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 suboptimals 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

<i>e1</i>	A pointer to a variable where the energy of molecule A will be written to
<i>e2</i>	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 /home/mescalini/ronny/WORK/ViennaRNA/H/convert_epars.h File Reference

Functions and definitions for energy parameter file format conversion.

Macros

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

Functions

- void [convert_parameter_file](#) (const char *iname, const char *oname, unsigned int options)

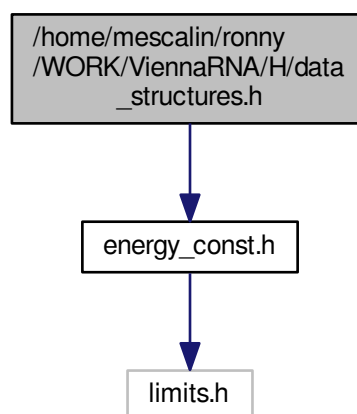
11.5.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

11.6 /home/mescal/ronny/WORK/ViennaRNA/H/data_structures.h File Reference

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

Include dependency graph for data_structures.h:



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



Data Structures

- struct [plist](#)
this datastructure is used as input parameter in functions of [PS_dot.h](#) and others
- struct [cpair](#)
this datastructure is used as input parameter in functions of [PS_dot.c](#)
- struct [COORDINATE](#)
this is a 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 [bondTEn](#)
Base pair with associated energy.
- struct [model_detailsT](#)
The data structure that contains the complete model details used throughout the calculations.
- struct [paramT](#)
The datastructure that contains temperature scaled energy parameters.
- struct [pf_paramT](#)
The datastructure that contains temperature scaled Boltzmann weights of the energy parameters.
- struct [PAIR](#)
Base pair data structure used in subopt.c.
- struct [INTERVAL](#)
Sequence interval stack element used in subopt.c.
- struct [SOLUTION](#)
Solution element from subopt.c.
- struct [cofoldF](#)
- struct [ConcEnt](#)
- struct [pairpro](#)
- struct [pair_info](#)
A base pair info structure.
- struct [move_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 beeing 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.

Macros

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

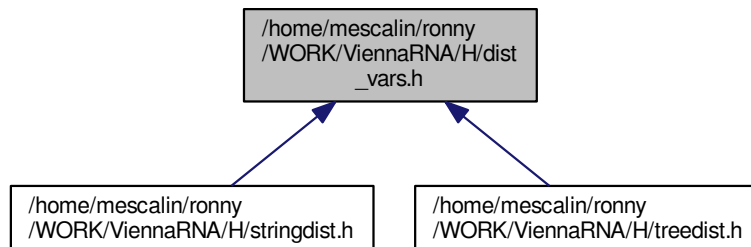
11.6.1 Detailed Description

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

11.7 /home/mescal/ronny/WORK/ViennaRNA/H/dist_vars.h File Reference

Global variables for Distance-Package.

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



Data Structures

- struct [Postorder_list](#)
- struct [Tree](#)
- struct [swString](#)

Variables

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

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

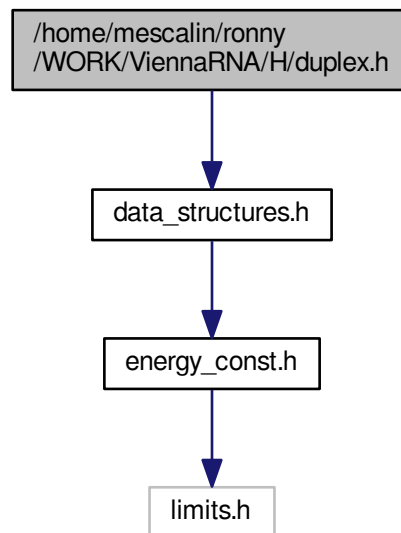
Specify the cost matrix to be used for distance calculations.

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

11.8 /home/mescal/ronny/WORK/ViennaRNA/H/duplex.h File Reference

Duplex folding function declarations...

Include dependency graph for duplex.h:



11.8.1 Detailed Description

Duplex folding function declarations...

11.9 /home/mescal/ronny/WORK/ViennaRNA/H/edit_cost.h File Reference

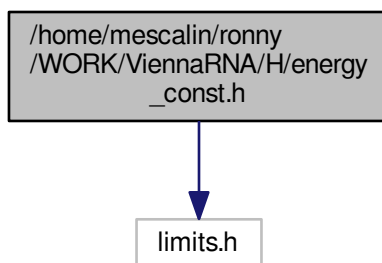
global variables for Edit Costs included by treedist.c and stringdist.c

11.9.1 Detailed Description

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

11.10 /home/mescal/ronny/WORK/ViennaRNA/H/energy_const.h File Reference

Include dependency graph for energy_const.h:



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



Macros

- `#define GASCONST 1.98717 /* in [cal/K] */`
- `#define K0 273.15`
- `#define INF 10000000 /* (INT_MAX/10) */`
- `#define FORBIDDEN 9999`
- `#define BONUS 10000`
- `#define NBPAIRS 7`
- `#define TURN 3`
- `#define MAXLOOP 30`

11.10.1 Detailed Description

energy constants

11.10.2 Macro Definition Documentation

11.10.2.1 `#define GASCONST 1.98717 /* in [cal/K] */`

The gas constant

11.10.2.2 `#define K0 273.15`

0 deg Celsius in Kelvin

11.10.2.3 `#define INF 10000000 /* (INT_MAX/10) */`

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

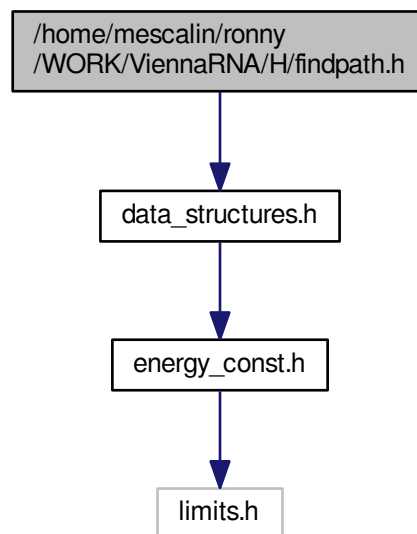
11.10.2.8 `#define MAXLOOP 30`

The maximum loop length

11.11 `/home/mescalin/ronny/WORK/ViennaRNA/H/findpath.h` File Reference

Compute direct refolding paths between two secondary structures.

Include dependency graph for findpath.h:



Functions

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

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

<i>seq</i>	RNA sequence
<i>struc1</i>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<i>struc2</i>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<i>max</i>	integer how many strutures 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 (serch only direct path)

Parameters

<i>seq</i>	RNA sequence
<i>s1</i>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<i>s2</i>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<i>maxkeep</i>	integer how many strutures 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.

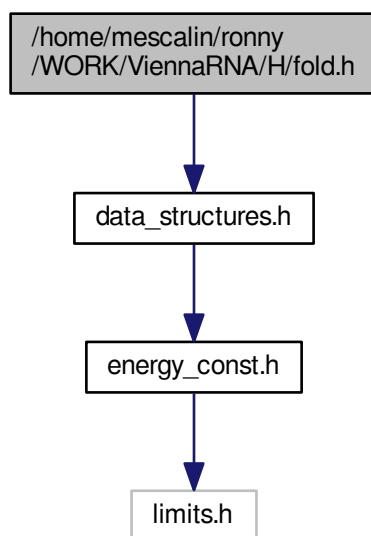
Parameters

<i>path</i>	pointer to memory to be freed
-------------	-------------------------------

11.12 /home/mescal/ronny/WORK/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

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation
<i>m1</i>	first coordinate of base pair
<i>m2</i>	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

<i>pt</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>m1</i>	first coordinate of base pair
<i>m2</i>	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

<i>p</i> table	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>i</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

<i>p</i>	A pointer to the plist that is to be created
<i>struc</i>	The secondary structure in dot-bracket notation
<i>pr</i>	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

<i>string</i>	RNA sequence
<i>structure</i>	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 * s1)`

Calculate the free energy of an already folded RNA

Note

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

Deprecated This function is deprecated and should not be used in future programs! Use [energy_of_structure_pt\(\)](#) instead!

See Also

[make_pair_table\(\)](#), [energy_of_structure\(\)](#)

Parameters

<i>string</i>	RNA sequence
<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	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

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation

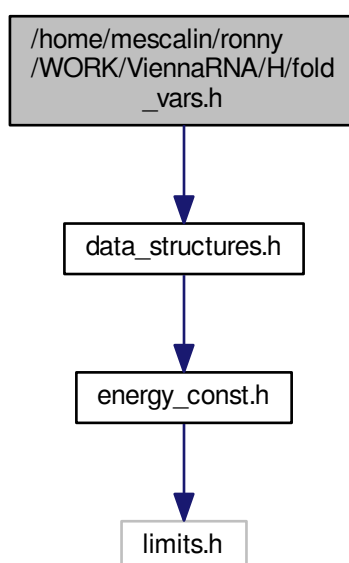
Returns

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

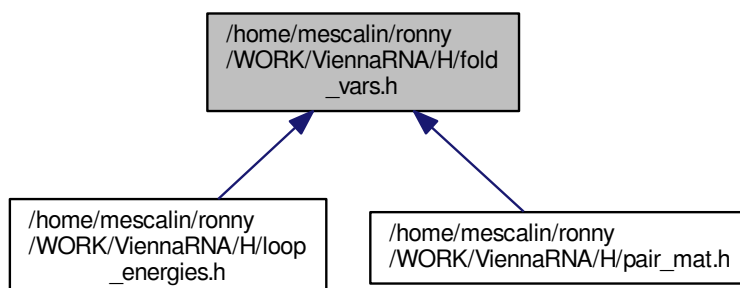
11.13 /home/mescal/ronny/WORK/ViennaRNA/H/fold_vars.h File Reference

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.
- int `no_closingGU`
GU allowed only inside stacks if set to 1.
- int `tetra_loop`
Include special stabilizing energies for some tri-, tetra- and hexa-loops;
- int `energy_set`
0 = BP; 1=any mit GC; 2=any mit AU-parameter
- int `circ`
backward compatibility variable.. this does not effect anything
- int `csv`
generate comma seperated output
- int `oldAliEn`
- int `ribo`
- char * `RibosumFile`
- char * `nonstandards`
contains allowed non standard base pairs
- double `temperature`
Rescale energy parameters to a temperature in degC.
- int `james_rule`
- int `logML`

- int `cut_point`
Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.
- `bondT * base_pair`
Contains a list of base pairs after a call to `fold()`.
- double * `pr`
A pointer to the base pair probability matrix.
- int * `iindx`
index array to move through `pr`.
- double `pf_scale`
A scaling factor used by `pf_fold()` to avoid overflows.
- int `do_backtrack`
do backtracking, i.e. compute secondary structures or base pair probabilities
- char `backtrack_type`
A backtrack array marker for `inverse_fold()`
- int `gquad`
Allow G-quadruplex formation.
- int `canonicalBPonly`

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

<code>md</code>	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 multi-loops. The option affects only mfe folding and energy evaluation (`fold()` and `energy_of_structure()`), as well as suboptimal folding (`subopt()`) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

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

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.

11.13.3.17 int do_backtrack

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

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

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.

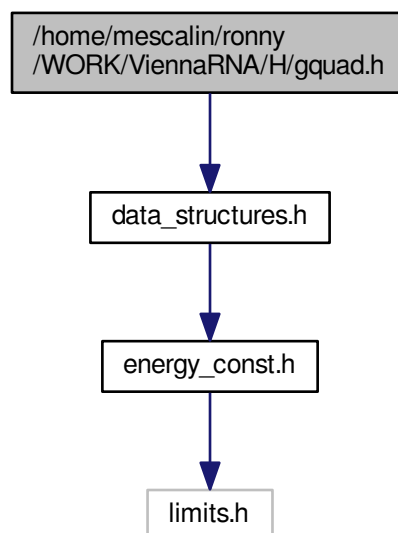
11.13.3.19 int canonicalBPonly

Do not use this variable, it will eventually be removed in one of the next versions

11.14 /home/mescalini/ronny/WORK/ViennaRNA/H/gquad.h File Reference

Various functions related to G-quadruplex computations.

Include dependency graph for gquad.h:



Functions

- int * [get_gquad_matrix](#) (short *S, [paramT](#) *P)
Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.
- int [parse_gquad](#) (const char *struc, int *L, int l[3])
- PRIVATE int [backtrack_GQuad_IntLoop](#) (int c, int i, int j, int type, short *S, int *ggg, int *index, int *p, int *q, [paramT](#) *P)
- PRIVATE int [backtrack_GQuad_IntLoop_L](#) (int c, int i, int j, int type, short *S, int **ggg, int maxdist, int *p, int *q, [paramT](#) *P)

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[jj]+i]`. To get the integer array `indx` see `get_jindx()`.

See Also

`get_jindx()`, `encode_sequence()`

Parameters

<i>S</i>	The encoded sequence
<i>P</i>	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

<i>c</i>	The total contribution the loop should resemble
<i>i</i>	position <i>i</i> of enclosing pair
<i>j</i>	position <i>j</i> of enclosing pair
<i>type</i>	base pair type of enclosing pair (must be reverse type)
<i>S</i>	integer encoded sequence
<i>ggg</i>	triangular matrix containing g-quadruplex contributions
<i>index</i>	the index for accessing the triangular matrix
<i>p</i>	here the 5' position of the gquad is stored
<i>q</i>	here the 3' position of the gquad is stored
<i>P</i>	the datastructure containing the precalculated contibutions

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

<i>c</i>	The total contribution the loop should resemble
<i>i</i>	position i of enclosing pair
<i>j</i>	position j of enclosing pair
<i>type</i>	base pair type of enclosing pair (must be reverse type)
<i>S</i>	integer encoded sequence
<i>ggg</i>	triangular matrix containing g-quadruplex contributions
<i>p</i>	here the 5' position of the gquad is stored
<i>q</i>	here the 3' position of the gquad is stored
<i>P</i>	the datastructure containing the precalculated contributions

Returns

1 on success, 0 if no gquad found

11.15 /home/mescalini/ronny/WORK/ViennaRNA/H/inverse.h File Reference

Inverse folding routines.

Functions

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

Variables

- char * [symbolset](#)
This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.
- float [final_cost](#)
- int [give_up](#)
- int [inv_verbose](#)

11.15.1 Detailed Description

Inverse folding routines.

11.16 /home/mescalini/ronny/WORK/ViennaRNA/H/Lfold.h File Reference

Predicting local MFE structures of large sequences.

Functions

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

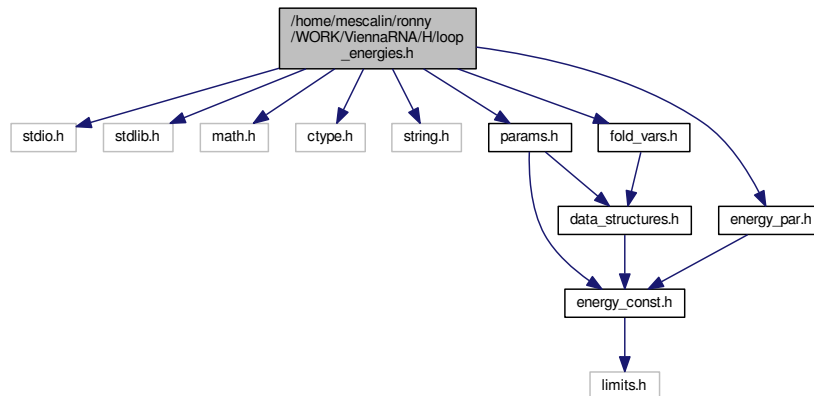
11.16.1 Detailed Description

Predicting local MFE structures of large sequences.

11.17 /home/mescal/ronny/WORK/ViennaRNA/H/loop_energies.h File Reference

Energy evaluation for MFE and partition function calculations.

Include dependency graph for loop_energies.h:



Functions

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

11.17.1 Detailed Description

Energy evaluation for MFE and partition function calculations. This file contains functions for the calculation of the free energy ΔG of a hairpin- [[E_Hairpin\(\)](#)] or interior-loop [[E_IntLoop\(\)](#)].

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

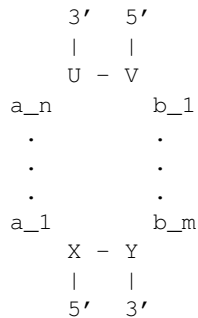
In case of computing the partition function, this file also supplies functions which return the Boltzmann weights $e^{-\Delta G/kT}$ for a hairpin- [[exp_E_Hairpin\(\)](#)] or interior-loop [[exp_E_IntLoop\(\)](#)].

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 ΔG 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

<i>n1</i>	The size of the 'left'-loop (number of unpaired nucleotides)
<i>n2</i>	The size of the 'right'-loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the interior loop
<i>type_2</i>	The pair type of the enclosed base pair
<i>si1</i>	The 5'-mismatching nucleotide of the closing pair
<i>sj1</i>	The 3'-mismatching nucleotide of the closing pair
<i>sp1</i>	The 3'-mismatching nucleotide of the enclosed pair
<i>sq1</i>	The 5'-mismatching nucleotide of the enclosed pair
<i>P</i>	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

<i>size</i>	The size of the loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the hairpin
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>string</i>	The sequence of the loop
<i>P</i>	The datastructure containing scaled energy parameters

Returns

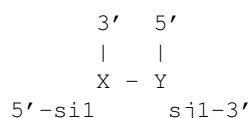
The Free energy of the Hairpin-loop in dcal/mol

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 sij or sj1 is a negative value, only 5' or 3' dangling end contributions are taken into account. To prohibit any of these mismatch contributions to be incorporated, just pass a negative number to both, si1 and sj1. In case the argument extLoop is 0, the returned energy contribution also includes the *internal-loop-penalty* of a multiloop stem with closing pair type.

See Also

E_MLstem()
E_ExtLoop()

Note

This function is threadsafe

Parameters

<i>type</i>	The pair type of the first base pair on the stem
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>extLoop</i>	A flag that indicates whether the contribution reflects the one of an exterior loop or not
<i>P</i>	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

<i>u</i>	The size of the loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the hairpin
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>string</i>	The sequence of the loop
<i>P</i>	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

Parameters

<i>u1</i>	The size of the 'left'-loop (number of unpaired nucleotides)
<i>u2</i>	The size of the 'right'-loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the interior loop
<i>type2</i>	The pair type of the enclosed base pair
<i>si1</i>	The 5'-mismatching nucleotide of the closing pair
<i>sj1</i>	The 3'-mismatching nucleotide of the closing pair
<i>sp1</i>	The 3'-mismatching nucleotide of the enclosed pair
<i>sq1</i>	The 5'-mismatching nucleotide of the enclosed pair
<i>P</i>	The datastructure containing scaled Boltzmann weights of the energy parameters

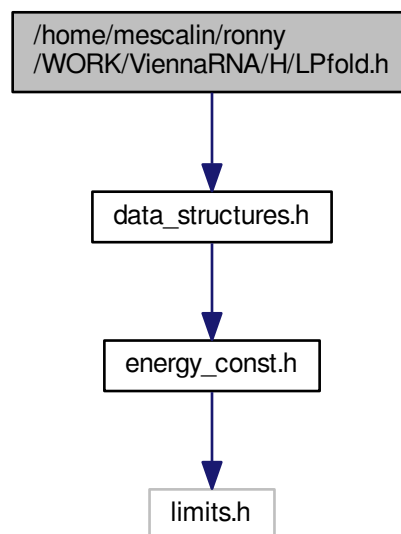
Returns

The Boltzmann weight of the Interior-loop

11.18 /home/mescal/ronny/WORK/ViennaRNA/H/LPfold.h File Reference

Function declarations of partition function variants of the Lfold algorithm.

Include dependency graph for LPfold.h:



Functions

- void [update_pf_paramsLP](#) (int length)
- [plist * pfl_fold](#) (char *sequence, int winSize, int pairSize, float cutoffb, double **pU, struct [plist](#) **dpp2, FILE *pUfp, FILE *spup)
Compute partition functions for locally stable secondary structures.
- [plist * pfl_fold_par](#) (char *sequence, int winSize, int pairSize, float cutoffb, double **pU, struct [plist](#) **dpp2, FILE *pUfp, FILE *spup, [pf_paramT](#) *parameters)
Compute partition functions for locally stable secondary structures.

- void [putoutpU_prob](#) (double **pU, int length, int ulength, FILE *fp, int energies)
Writes the unpaired probabilities (pU) or opening energies into a file.
- void [putoutpU_prob_bin](#) (double **pU, int length, int ulength, FILE *fp, int energies)
Writes the unpaired probabilities (pU) or opening energies into a binary file.
- void [init_pf_foldLP](#) (int length)

11.18.1 Detailed Description

Function declarations of partition function variants of the Lfold algorithm.

11.18.2 Function Documentation

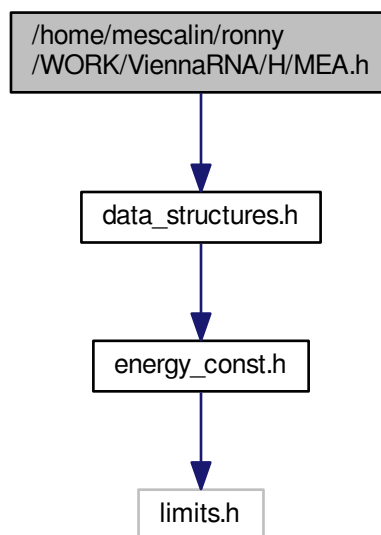
11.18.2.1 void [init_pf_foldLP](#) (int *length*)

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 /home/mescal/ronny/WORK/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 γ result in more base pairs of lower probability and thus higher sensitivity. Low values of γ result in structures containing only highly likely pairs (high specificity). The code of the MEA function also demonstrates the use of sparse dynamic programming scheme to reduce the time and memory complexity of folding.

11.20 /home/mescal/ronny/WORK/ViennaRNA/H/mm.h File Reference

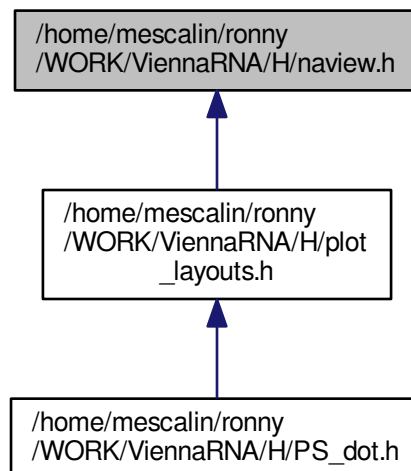
Several Maximum Matching implementations.

11.20.1 Detailed Description

Several Maximum Matching implementations. This file contains the declarations for several maximum matching implementations

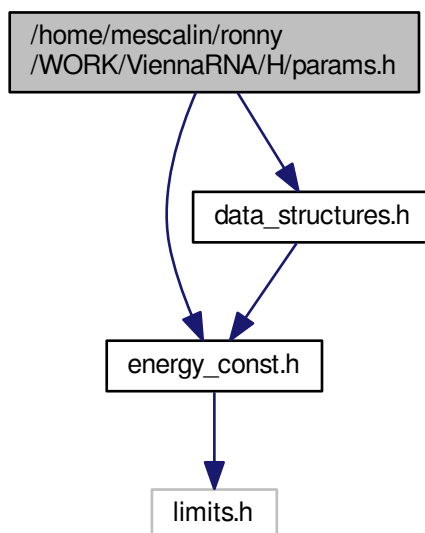
11.21 /home/mescal/ronny/WORK/ViennaRNA/H/naview.h File Reference

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

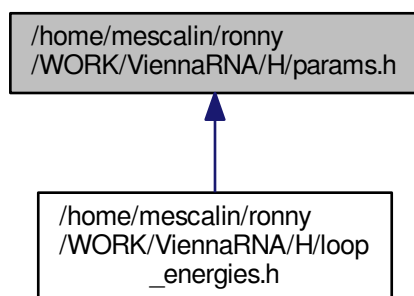


11.22 /home/mescal/ronny/WORK/ViennaRNA/H/params.h File Reference

Include dependency graph for params.h:



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



Functions

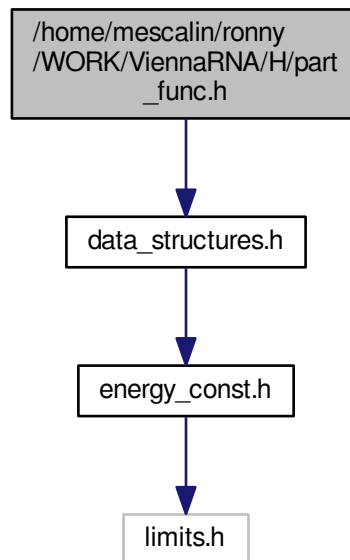
- `paramT * scale_parameters` (void)
Get precomputed energy contributions for all the known loop types.
- `paramT * get_scaled_parameters` (double `temperature`, `model_detailsT` md)
Get precomputed energy contributions for all the known loop types.
- `pf_paramT * get_scaled_pf_parameters` (void)

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

11.23 /home/mescal/ronny/WORK/ViennaRNA/H/part_func.h File Reference

Partition function of single RNA sequences.

Include dependency graph for `part_func.h`:



Functions

- float `pf_fold_par` (const char *`sequence`, char *`structure`, `pf_paramT` *`parameters`, int `calculate_bppm`, int `is_constrained`, int `is_circular`)
Compute the partition function Q for a given RNA sequence.
- float `pf_fold` (const char *`sequence`, char *`structure`)
Compute the partition function Q of an RNA sequence.
- float `pf_circ_fold` (const char *`sequence`, char *`structure`)
Compute the partition function of a circular RNA sequence.

- char * [pbacktrack](#) (char *sequence)
Sample a secondary structure from the Boltzmann ensemble according its probability
- char * [pbacktrack_circ](#) (char *sequence)
Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.
- void [free_pf_arrays](#) (void)
Free arrays for the partition function recursions.
- void [update_pf_params](#) (int length)
Recalculate energy parameters.
- void [update_pf_params_par](#) (int length, [pf_paramT](#) *parameters)
Recalculate energy parameters.
- double * [export_bppm](#) (void)
Get a pointer to the base pair probability array
Accessing the base pair probabilities for a pair (i,j) is achieved by.
- void [assign_plist_from_pr](#) ([plist](#) **pl, double *probs, int length, double cutoff)
Create a plist from a probability matrix.
- int [get_pf_arrays](#) (short **S_p, short **S1_p, char **ptype_p, double **qb_p, double **qm_p, double **q1k_p, double **qln_p)
Get the pointers to (almost) all 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.

11.23.1 Detailed Description

Partition function of single RNA sequences. This file includes (almost) all function declarations within the **RNAlib** that are related to Partition 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

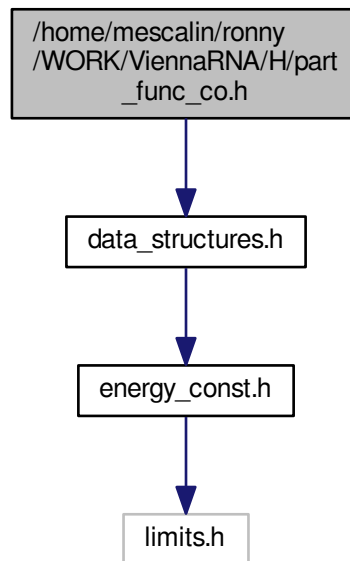
11.23.2.5 double expHairpinEnergy (int *u*, int *type*, short *si1*, short *sj1*, const char * *string*)

Deprecated Use [exp_E_Hairpin\(\)](#) from [loop_energies.h](#) instead

11.24 /home/mescal/ronny/WORK/ViennaRNA/H/part_func_co.h File Reference

Partition function for two RNA sequences.

Include dependency graph for `part_func_co.h`:



Functions

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

Variables

- int [mirnatog](#)

Toggles no intrabp in 2nd mol.

- double [F_monomer](#) [2]

Free energies of the two monomers.

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 so far and so away.

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)`

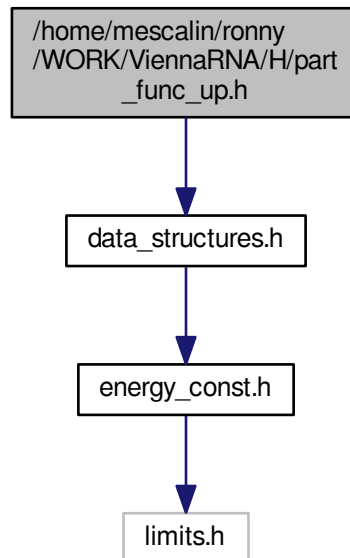
DO NOT USE THIS FUNCTION ANYMORE

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

11.25 /home/mescalini/ronny/WORK/ViennaRNA/H/part_func_up.h File Reference

Partition Function Cofolding as stepwise process.

Include dependency graph for `part_func_up.h`:



Functions

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

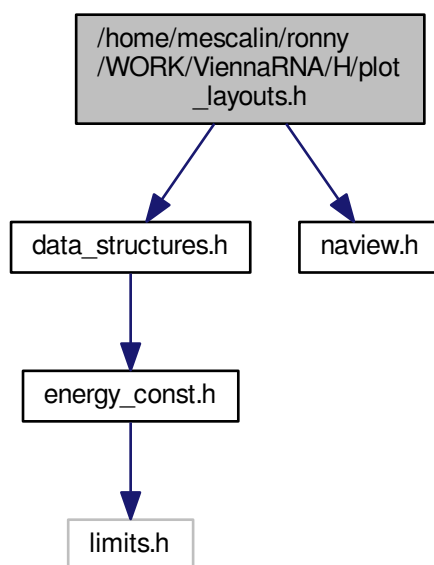
11.25.1 Detailed Description

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.

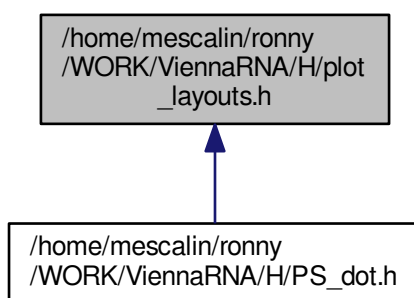
11.26 /home/mescalini/ronny/WORK/ViennaRNA/H/plot_layouts.h File Reference

Secondary structure plot layout algorithms.

Include dependency graph for plot_layouts.h:



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



Macros

- `#define VRNA_PLOT_TYPE_SIMPLE 0`
Definition of Plot type simple
- `#define VRNA_PLOT_TYPE_NAVIEW 1`
Definition of Plot type Naview
- `#define VRNA_PLOT_TYPE_CIRCULAR 2`
Definition of Plot type Circular

Functions

- int [simple_xy_coordinates](#) (short *pair_table, float *X, float *Y)
Calculate nucleotide coordinates for secondary structure plot the Simple way
- int [simple_circplot_coordinates](#) (short *pair_table, float *x, float *y)
Calculate nucleotide coordinates for Circular Plot

Variables

- int [rna_plot_type](#)
Switch for changing the secondary structure layout algorithm.

11.26.1 Detailed Description

Secondary structure plot layout algorithms. c Ronny Lorenz The ViennaRNA Package

11.26.2 Macro Definition 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

<i>pair_table</i>	The pair table of the secondary structure
<i>X</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>Y</i>	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

<i>pair_table</i>	The pair table of the secondary structure
<i>x</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>y</i>	a pointer to an array with enough allocated space to hold the y coordinates

Returns

length of sequence on success, 0 otherwise

11.26.4 Variable Documentation

11.26.4.1 `int rna_plot_type`

Switch for changing the secondary structure layout algorithm.

Current possibility are 0 for a simple radial drawing or 1 for the modified radial drawing taken from the *naview* program of Brucocoleri & Heinrich (1988).

Note

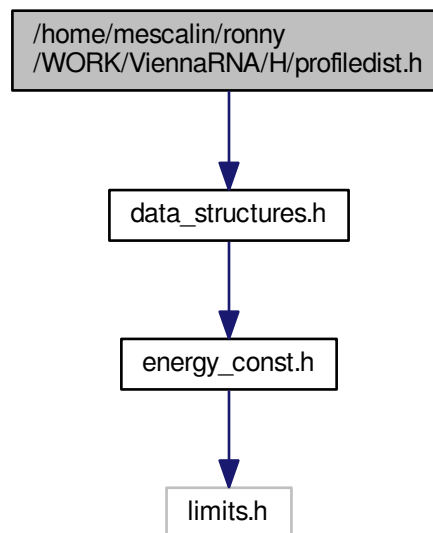
To provide thread safety please do not rely on this global variable in future implementations but pass a plot type flag directly to the function that decides which layout algorithm it may use!

See Also

[VRNA_PLOT_TYPE_SIMPLE](#), [VRNA_PLOT_TYPE_NAVIEW](#), [VRNA_PLOT_TYPE_CIRCULAR](#)

11.27 /home/mescalini/ronny/WORK/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 Function Documentation

11.27.1.1 `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

11.27.1.2 `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

<i>bppm</i>	A pointer to the base pair probability matrix
<i>length</i>	The length of the sequence

Returns

The bp profile

11.27.1.3 `void free_profile (float * T)`

free space allocated in `Make_bp_profile`

Backward compatibility only. You can just use plain `free()`

11.27.1.4 `float* Make_bp_profile (int length)`

Note

This function is NOT threadsafe

See Also

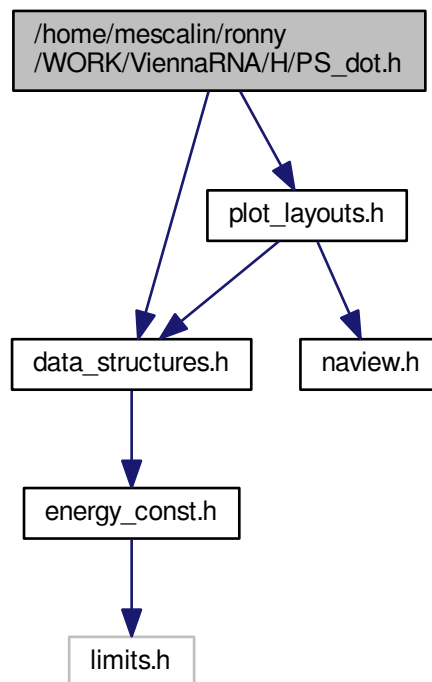
[Make_bp_profile_bppm\(\)](#)

Deprecated This function is deprecated and will be removed soon! See [Make_bp_profile_bppm\(\)](#) for a replacement

11.28 /home/mescalini/ronny/WORK/ViennaRNA/H/PS_dot.h File Reference

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Include dependency graph for PS_dot.h:



Functions

- `int PS_rna_plot (char *string, char *structure, char *file)`
Produce a secondary structure graph in PostScript and write it to 'filename'.
- `int PS_rna_plot_a (char *string, char *structure, char *file, char *pre, char *post)`
Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.
- `int gmlRNA (char *string, char *structure, char *ssfile, char option)`
Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.
- `int ssv_rna_plot (char *string, char *structure, char *ssfile)`
Produce a secondary structure graph in SStructView format.
- `int svg_rna_plot (char *string, char *structure, char *ssfile)`
Produce a secondary structure plot in SVG format and write it to a file.
- `int xrna_plot (char *string, char *structure, char *ssfile)`
Produce a secondary structure plot for further editing in XRNA.
- `int PS_dot_plot_list (char *seq, char *filename, plist *pl, plist *mf, char *comment)`
Produce a postscript dot-plot from two pair lists.
- `int aliPS_color_aln (const char *structure, const char *filename, const char *seqs[], const char *names[])`
- `int PS_dot_plot (char *string, char *file)`
Produce postscript dot-plot.

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

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>file</i>	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

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>file</i>	The filename of the postscript output
<i>pre</i>	PostScript code to appear before the secondary structure plot
<i>post</i>	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 contain coordinates for an initial layout of the graph.

Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the gml output
<i>option</i>	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

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	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.

Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	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

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	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

<i>seq</i>	The RNA sequence
<i>filename</i>	A filename for the postscript output
<i>pl</i>	The base pair probability pairlist
<i>mf</i>	The mfe secondary structure pairlist
<i>comment</i>	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 THREADSAFE

Deprecated This function is deprecated and will be removed soon! Use [PS_dot_plot_list\(\)](#) instead!

11.29 /home/mescal/ronny/WORK/ViennaRNA/H/read_epars.h File Reference

Functions

- void [read_parameter_file](#) (const char fname[])
Read energy parameters from a file.
- void [write_parameter_file](#) (const char fname[])
Write energy parameters to a file.

11.30 /home/mescal/ronny/WORK/ViennaRNA/H/RNAstruct.h File Reference

Parsing and Coarse Graining of Structures.

Functions

- char * [b2HIT](#) (const char *structure)
Converts the full structure from bracket notation to the HIT notation including root.
- char * [b2C](#) (const char *structure)
Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.
- char * [b2Shapiro](#) (const char *structure)
Converts the full structure from bracket notation to the weighted coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.
- char * [add_root](#) (const char *structure)
Adds a root to an un-rooted tree in any except bracket notation.
- char * [expand_Shapiro](#) (const char *coarse)
Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from [b2C\(\)](#).
- char * [expand_Full](#) (const char *structure)
Convert the full structure from bracket notation to the expanded notation including root.
- char * [unexpand_Full](#) (const char *ffull)
Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.
- char * [unweight](#) (const char *wcoarse)
Strip weights from any weighted tree.
- void [unexpand_aligned_F](#) (char *align[2])
Converts two aligned structures in expanded notation.
- void [parse_structure](#) (const char *structure)
Collects a statistic of structure elements of the full structure in bracket notation.

Variables

- int [loop_size](#) [STRUC]
contains a list of all loop sizes. `loop_size[0]` contains the number of external bases.
- int [helix_size](#) [STRUC]
contains a list of all stack sizes.
- int [loop_degree](#) [STRUC]
contains the corresponding list of loop degrees.
- int [loops](#)
contains the number of loops (and therefore of stacks).
- int [unpaired](#)
contains the number of unpaired bases.
- int [pairs](#)
contains the number of base pairs in the last parsed structure.

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)(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

<i>structure</i>	
------------------	--

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

<i>structure</i>	
------------------	--

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

<i>structure</i>	
------------------	--

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

<i>structure</i>	
------------------	--

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

<i>coarse</i>	
---------------	--

Returns**11.30.2.6 char* expand_Full (const char * *structure*)**

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

Parameters

<i>structure</i>	
------------------	--

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

<i>ffull</i>	
--------------	--

Returns**11.30.2.8 char* unweight (const char * *wcoarse*)**

Strip weights from any weighted tree.

Parameters

<i>wcoarse</i>	
----------------	--

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

<i>align</i>	
--------------	--

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

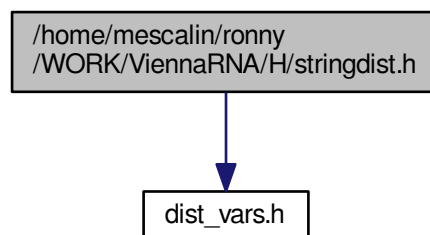
<i>structure</i>	
------------------	--

Returns

11.31 /home/mescal/ronny/WORK/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

<i>string</i>	
---------------	--

Returns

11.31.2.2 `float string_edit_distance (swString * T1, swString * T2)`

Calculate the string edit distance of *T1* and *T2*.

Parameters

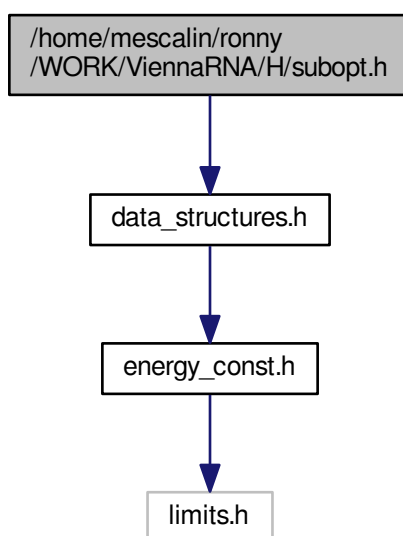
<i>T1</i>	
<i>T2</i>	

Returns

11.32 `/home/mescal/ronny/WORK/ViennaRNA/H/subopt.h` File Reference

RNAsubopt and density of states declarations.

Include dependency graph for subopt.h:



Functions

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

Variables

- int **subopt_sorted**
Sort output by energy.
- double **print_energy**
printing threshold for use with logML
- int **density_of_states** [MAXDOS+1]
The Density of States.

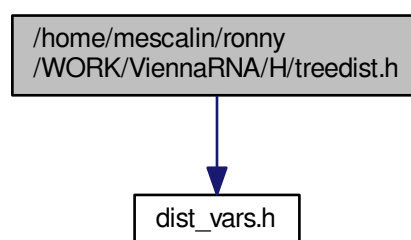
11.32.1 Detailed Description

RNAsubopt and density of states declarations.

11.33 /home/mescal/ronny/WORK/ViennaRNA/H/treedist.h File Reference

Functions for **Tree** Edit Distances.

Include dependency graph for treedist.h:



Functions

- **Tree** * **make_tree** (char *struc)
*Constructs a **Tree** (essentially the postorder list) of the structure 'struc', for use in **tree_edit_distance**().*
- float **tree_edit_distance** (**Tree** *T1, **Tree** *T2)
Calculates the edit distance of the two trees.
- void **print_tree** (**Tree** *t)

Print a tree (mainly for debugging)

- void `free_tree` (`Tree *t`)

Free the memory allocated for `Tree t`.

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

<code>struc</code>	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

<code>T1</code>	
<code>T2</code>	

Returns

11.33.2.3 `void free_tree (Tree * t)`

Free the memory allocated for `Tree t`.

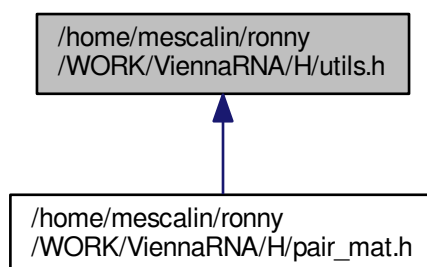
Parameters

<code>t</code>	
----------------	--

11.34 /home/mescalin/ronny/WORK/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:



Macros

- `#define VRNA_INPUT_ERROR 1U`
- `#define VRNA_INPUT_QUIT 2U`
- `#define VRNA_INPUT_MISC 4U`
- `#define VRNA_INPUT_FASTA_HEADER 8U`
- `#define VRNA_INPUT_SEQUENCE 16U`
- `#define VRNA_INPUT_CONSTRAINT 32U`
- `#define VRNA_INPUT_NO_TRUNCATION 256U`
- `#define VRNA_INPUT_NO_REST 512U`
- `#define VRNA_INPUT_NO_SPAN 1024U`
- `#define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U`
- `#define VRNA_INPUT_BLANK_LINE 4096U`
- `#define VRNA_INPUT_NOSKIP_COMMENTS 128U`
- `#define VRNA_INPUT_COMMENT 8192U`
- `#define VRNA_CONSTRAINT_PIPE 1U`
- `#define VRNA_CONSTRAINT_DOT 2U`
- `#define VRNA_CONSTRAINT_X 4U`
- `#define VRNA_CONSTRAINT_ANG_BRACK 8U`
- `#define VRNA_CONSTRAINT_RND_BRACK 16U`
- `#define VRNA_CONSTRAINT_MULTILINE 32U`
- `#define VRNA_CONSTRAINT_NO_HEADER 64U`
- `#define VRNA_CONSTRAINT_ALL 128U`
- `#define VRNA_CONSTRAINT_G 256U`
- `#define VRNA_OPTION_MULTILINE 32U`
- `#define MIN2(A, B) ((A) < (B) ? (A) : (B))`
- `#define MAX2(A, B) ((A) > (B) ? (A) : (B))`
- `#define MIN3(A, B, C) (MIN2((MIN2((A),(B))), (C)))`
- `#define MAX3(A, B, C) (MAX2((MAX2((A),(B))), (C)))`
- `#define XSTR(s) STR(s)`
- `#define STR(s) #s`
- `#define FILENAME_MAX_LENGTH 80`
Maximum length of filenames that are generated by our programs.
- `#define FILENAME_ID_LENGTH 42`
Maximum length of id taken from fasta header for filename generation.

Functions

- void * [space](#) (unsigned size)
Allocate space safely.
- void * [xrealloc](#) (void *p, unsigned size)
Reallocate space safely.
- void [nrerror](#) (const char message[])
Die with an error message.
- void [warn_user](#) (const char message[])
Print a warning message.
- void [init_rand](#) (void)
Make random number seeds.
- double [urn](#) (void)
get a random number from [0..1]
- int [int_urn](#) (int from, int to)
Generates a pseudo random integer in a specified range.
- char * [time_stamp](#) (void)
Get a timestamp.
- char * [random_string](#) (int l, const char symbols[])
Create a random string using characters from a specified symbol set.
- int [hamming](#) (const char *s1, const char *s2)
Calculate hamming distance between two sequences.
- int [hamming_bound](#) (const char *s1, const char *s2, int n)
Calculate hamming distance between two sequences up to a specified length.
- char * [get_line](#) (FILE *fp)
Read a line of arbitrary length from a stream.
- unsigned int [get_input_line](#) (char **string, unsigned int options)
- unsigned int [read_record](#) (char **header, char **sequence, char ***rest, unsigned int options)
Get a data record from stdin.
- char * [pack_structure](#) (const char *struc)
Pack secondary secondary structure, 5:1 compression using base 3 encoding.
- char * [unpack_structure](#) (const char *packed)
Unpack secondary structure previously packed with [pack_structure\(\)](#)
- short * [make_pair_table](#) (const char *structure)
Create a pair table of a secondary structure.
- short * [copy_pair_table](#) (const short *pt)
Get an exact copy of a pair table.
- short * [alimake_pair_table](#) (const char *structure)
- short * [make_pair_table_snoop](#) (const char *structure)
- int * [make_loop_index_pt](#) (short *pt)
Compute the "base pair" distance between two secondary structures s1 and s2.
- void [print_tty_input_seq](#) (void)
Print a line to stdout that asks for an input sequence.
- void [print_tty_input_seq_str](#) (const char *s)
Print a line with a user defined string and a ruler to stdout.
- void [print_tty_constraint_full](#) (void)
Print structure constraint characters to stdout (full constraint support)
- void [print_tty_constraint](#) (unsigned int option)
Print structure constraint characters to stdout. (constraint support is specified by option parameter)
- void [str_DNA2RNA](#) (char *sequence)
Convert a DNA input sequence to RNA alphabet.

- void `str_uppercase` (char *sequence)
Convert an input sequence to uppercase.
- int * `get_iindx` (unsigned int length)
Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.
- int * `get_indx` (unsigned int length)
Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.
- void `constrain_ptypes` (const char *constraint, unsigned int length, char *ptype, int *BP, int min_loop_size, unsigned int idx_type)
Insert constraining pair types according to constraint structure string.

Variables

- unsigned short `xsubi` [3]
Current 48 bit random number.

11.34.1 Detailed Description

Various utility- and helper-functions used throughout the Vienna RNA package.

11.34.2 Macro Definition 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)

11.34.2.17 `#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 file-names 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

<i>size</i>	The size of the memory to be allocated in bytes
-------------	-------------------------------------------------

Returns

A pointer to the allocated memory

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

Reallocate space safely.

Parameters

<i>p</i>	A pointer to the memory region to be reallocated
<i>size</i>	The size of the memory to be allocated in bytes

Returns

A pointer to the newly allocated memory

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

Die with an error message.

See Also

[warn_user\(\)](#)

Parameters

<i>message</i>	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

<i>message</i>	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

<i>from</i>	The first number in range
<i>to</i>	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

<i>l</i>	The length of the sequence
<i>symbols</i>	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

<i>s1</i>	The first sequence
<i>s2</i>	The second sequence

Returns

The hamming distance between s1 and s2

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

<i>s1</i>	The first sequence
<i>s2</i>	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

<i>fp</i>	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' safely 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

<i>string</i>	A pointer to the character array that contains the line read
<i>options</i>	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]);
}
```

In the example above, the while loop will be terminated when `read_record()` returns either an error or a user initiated quit request.\n
 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.

Do not forget to free the memory occupied by header, sequence and rest!

Parameters

<i>header</i>	A pointer which will be set such that it points to the header of the record
<i>sequence</i>	A pointer which will be set such that it points to the sequence of the record
<i>rest</i>	A pointer which will be set such that it points to an array of lines which also belong to the record
<i>options</i>	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

<i>struc</i>	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

<i>packed</i>	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

<i>structure</i>	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

<i>pt</i>	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

<i>str1</i>	First structure in dot-bracket notation
<i>str2</i>	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

<i>s</i>	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

<i>option</i>	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

<i>sequence</i>	The sequence to be converted
-----------------	------------------------------

11.34.3.25 void str_uppercase (char * *sequence*)

Convert an input sequence to uppercase.

Parameters

<i>sequence</i>	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) ~ 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

<i>length</i>	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) ~ 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

<i>length</i>	The length of the RNA sequence
---------------	--------------------------------

Returns

The mapper array

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

<i>constraint</i>	The structure constraint string
<i>length</i>	The actual length of the sequence (constraint may be shorter)
<i>ptype</i>	A pointer to the basepair type array
<i>min_loop_size</i>	The minimal loop size (usually TURN)
<i>idx_type</i>	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 /home/mescalini/ronny/WORK/ViennaRNA/lib/1.8.4_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 Vienna-RNAPackage 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 /home/mescalini/ronny/WORK/ViennaRNA/lib/1.8.4_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.

Bibliography

- [1] S.H. Bernhart, I.L. Hofacker, S. Will, A.R. Gruber, and P.F. Stadler. RNAalifold: Improved consensus structure prediction for RNA alignments. *BMC bioinformatics*, 9(1):474, 2008.
- [2] S.H. Bernhart, H. Tafer, U. Mückstein, C. Flamm, P.F. Stadler, and I.L. Hofacker. Partition function and base pairing probabilities of RNA heterodimers. *Algorithms for Molecular Biology*, 1(1):3, 2006.
- [3] W. Fontana, P.F. Stadler, E.G. Bornberg-Bauer, T. Griesmacher, I.L. Hofacker, M. Tacker, P. Tarazona, E.D. Weinberger, and P. Schuster. RNA folding and combinatorial landscapes. *Physical review E*, 47(3):2083, 1993.
- [4] I.L. Hofacker, M. Fekete, and P.F. Stadler. Secondary structure prediction for aligned RNA sequences. *Journal of molecular biology*, 319(5):1059–1066, 2002.
- [5] I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994.
- [6] I.L. Hofacker and P.F. Stadler. Memory efficient folding algorithms for circular RNA secondary structures. *Bioinformatics*, 22(10):1172–1176, 2006.
- [7] Ronny Lorenz, Stephan H. Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011.
- [8] Ronny Lorenz, Christoph Flamm, and Ivo L. Hofacker. 2d projections of RNA folding landscapes. In Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, and Peter F. Stadler, editors, *German Conference on Bioinformatics 2009*, volume 157 of *Lecture Notes in Informatics*, pages 11–20, Bonn, September 2009. Gesellschaft f. Informatik.
- [9] D.H. Mathews, M.D. Disney, J.L. Childs, S.J. Schroeder, M. Zuker, and D.H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287, 2004.
- [10] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.
- [11] B.A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer applications in the biosciences: CABIOS*, 4(3):387–393, 1988.
- [12] D.H. Turner and D.H. Mathews. NNDB: The nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Research*, 38(suppl 1):D280–D282, 2010.
- [13] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.

Index

/home/mescalin/ronny/WORK/ViennaRNA/H/2Dfold.h, [107](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/2Dpfold.h, [108](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/LPfold.h, [139](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/Lfold.h, [133](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/MEA.h, [140](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/PS_dot.h, [153](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/RN-Astruct.h, [157](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/alifold.h, [108](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/cofold.h, [110](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/convert_-epars.h, [112](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/data_-structures.h, [113](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/dist_vars.-h, [115](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/duplex.h, [116](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/edit_cost.-h, [116](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/energy_-const.h, [117](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/findpath.h, [118](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/fold.h, [120](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/fold_vars.-h, [126](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/gquad.h, [131](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/inverse.h, [133](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/loop_-energies.h, [134](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/mm.h, [141](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/naview.h, [141](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/params.h, [142](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/part_-func.h, [143](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/part_func_-co.h, [145](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/part_func_-up.h, [147](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/plot_-layouts.h, [148](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/profiledist.-h, [152](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/read_-epars.h, [157](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/stringdist.-h, [161](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/subopt.h, [162](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/treedist.h, [163](#)
/home/mescalin/ronny/WORK/ViennaRNA/H/utils.h, [164](#)
/home/mescalin/ronny/WORK/ViennaRNA/lib/1.8.4_-epars.h, [178](#)
/home/mescalin/ronny/WORK/ViennaRNA/lib/1.8.4_-intloops.h, [178](#)
add_root
 RNAstruct.h, [159](#)
aliLfold
 Local MFE consensus structures for Sequence Alignments, [61](#)
aliPS_color_aln
 PS_dot.h, [157](#)
alifold
 MFE Consensus Structures for Sequence Alignment(s), [52](#)
alifold.h
 update_alifold_params, [110](#)
alimake_pair_table
 utils.h, [175](#)
alipbacktrack
 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), [56](#)
alipf_circ_fold
 Partition Function and Base Pair Probabilities for Sequence Alignment(s), [55](#)
alipf_fold
 Partition Function and Base Pair Probabilities for Sequence Alignment(s), [54](#)
alipf_fold_par
 Partition Function and Base Pair Probabilities for Sequence Alignment(s), [54](#)
alloc_sequence_arrays

- Predicting Consensus Structures from Alignment(s), [50](#)
- alpha
 - pf_paramT, [98](#)
- assign_plist_from_db
 - fold.h, [124](#)
- assign_plist_from_pr
 - Calculating Partition Functions and Pair Probabilities, [27](#)
- b2C
 - RNAstruct.h, [159](#)
- b2HIT
 - RNAstruct.h, [159](#)
- b2Shapiro
 - RNAstruct.h, [159](#)
- BONUS
 - energy_const.h, [118](#)
- backtrack_GQuad_IntLoop
 - gquad.h, [132](#)
- backtrack_GQuad_IntLoop_L
 - gquad.h, [132](#)
- backtrack_type
 - fold_vars.h, [130](#)
- base_pair
 - fold_vars.h, [130](#)
- bondT, [89](#)
- bondTEn, [89](#)
- COORDINATE, [90](#)
- Calculate Partition Functions of a Distance Based Partitioning, [82](#)
 - destroy_TwoDpfold_variables, [83](#)
 - get_TwoDpfold_variables, [82](#)
 - get_TwoDpfold_variables_from_MFE, [83](#)
 - TwoDpfoldList, [83](#)
- Calculate Secondary Structures of two RNAs upon Dimerization, [39](#)
- Calculating MFE representatives of a Distance Based Partitioning, [79](#)
 - destroy_TwoDfold_variables, [80](#)
 - get_TwoDfold_variables, [79](#)
 - TwoDfold_backtrack_f5, [81](#)
 - TwoDfoldList, [80](#)
- Calculating Minimum Free Energy (MFE) Structures, [20](#)
 - circfold, [22](#)
 - fold, [21](#)
 - fold_par, [21](#)
- Calculating Partition Functions and Pair Probabilities, [23](#)
 - assign_plist_from_pr, [27](#)
 - export_bppm, [27](#)
 - free_pf_arrays, [27](#)
 - get_pf_arrays, [28](#)
 - mean_bp_distance, [28](#)
 - mean_bp_distance_pr, [29](#)
 - pf_circ_fold, [26](#)
 - pf_fold, [25](#)
 - pf_fold_par, [24](#)
 - update_pf_params, [27](#)
- canonicalBPonly
 - fold_vars.h, [131](#)
- centroid
 - part_func.h, [145](#)
- Change and Precalculate Energy Parameter Sets and Boltzmann Factors, [62](#)
 - get_boltzmann_factor_copy, [64](#)
 - get_boltzmann_factors, [63](#)
 - get_scaled_parameters, [63](#)
 - get_scaled_pf_parameters, [63](#)
 - scale_parameters, [63](#)
- circalifold
 - MFE Consensus Structures for Sequence Alignment(s), [52](#)
- circfold
 - Calculating Minimum Free Energy (MFE) Structures, [22](#)
- Classified Dynamic Programming, [77](#)
- co_pf_fold
 - Partition Function for two hybridized Sequences, [43](#)
- co_pf_fold_par
 - Partition Function for two hybridized Sequences, [43](#)
- cofold
 - MFE Structures of two hybridized Sequences, [40](#)
- cofold.h
 - get_monomere_mfes, [112](#)
 - initialize_cofold, [112](#)
- cofoldF, [89](#)
- Compute the centroid structure, [31](#)
 - get_centroid_struct_pl, [31](#)
 - get_centroid_struct_pr, [31](#)
- Compute the Density of States, [87](#)
 - density_of_states, [87](#)
- Compute the structure with maximum expected accuracy (MEA), [30](#)
- compute_probabilities
 - Partition Function for two hybridized Sequences, [45](#)
- ConcEnt, [90](#)
- constrain, [90](#)
- constrain_ptypes
 - utils.h, [177](#)
- convert_parameter_file
 - Converting energy parameter files, [69](#)
- Converting energy parameter files, [67](#)
 - convert_parameter_file, [69](#)
- copy_pair_table
 - utils.h, [175](#)
- cost_matrix
 - dist_vars.h, [115](#)
- cpair, [90](#)
- cut_point
 - fold_vars.h, [130](#)
- cv_fact
 - Predicting Consensus Structures from Alignment(s), [51](#)
- dangles
 - fold_vars.h, [128](#)
 - model_detailsT, [94](#)

- density_of_states
 - Compute the Density of States, [87](#)
- destroy_TwoDfold_variables
 - Calculating MFE representatives of a Distance Based Partitioning, [80](#)
- destroy_TwoDpfold_variables
 - Calculate Partition Functions of a Distance Based Partitioning, [83](#)
- dist_vars.h
 - cost_matrix, [115](#)
 - edit_backtrack, [115](#)
- Distance based partitioning of the Secondary Structure Space, [78](#)
- do_backtrack
 - fold_vars.h, [130](#)
- dupVar, [91](#)
- duplexT, [91](#)
- E_Hairpin
 - loop_energies.h, [136](#)
- E_IntLoop
 - loop_energies.h, [135](#)
- E_Stem
 - loop_energies.h, [136](#)
- edit_backtrack
 - dist_vars.h, [115](#)
- encode.ali.sequence
 - Predicting Consensus Structures from Alignment(s), [49](#)
- Energy evaluation, [71](#)
 - energy_of_circ_struct_par, [73](#)
 - energy_of_circ_structure, [72](#)
 - energy_of_struct_par, [72](#)
 - energy_of_struct_pt_par, [74](#)
 - energy_of_structure, [71](#)
 - energy_of_structure_pt, [73](#)
- energy_const.h
 - BONUS, [118](#)
 - FORBIDDEN, [118](#)
 - GASCONST, [117](#)
 - INF, [117](#)
 - K0, [117](#)
 - MAXLOOP, [118](#)
 - NBPAIRS, [118](#)
 - TURN, [118](#)
- energy_of_alistruct
 - Predicting Consensus Structures from Alignment(s), [49](#)
- energy_of_circ_struct
 - fold.h, [125](#)
- energy_of_circ_struct_par
 - Energy evaluation, [73](#)
- energy_of_circ_structure
 - Energy evaluation, [72](#)
- energy_of_move
 - fold.h, [123](#)
- energy_of_move_pt
 - fold.h, [123](#)
- energy_of_struct
 - fold.h, [124](#)
- energy_of_struct_par
 - Energy evaluation, [72](#)
- energy_of_struct_pt
 - fold.h, [125](#)
- energy_of_struct_pt_par
 - Energy evaluation, [74](#)
- energy_of_structure
 - Energy evaluation, [71](#)
- energy_of_structure_pt
 - Energy evaluation, [73](#)
- energy_set
 - fold_vars.h, [129](#)
- Enumerating Suboptimal Structures, [33](#)
- exp_E_Hairpin
 - loop_energies.h, [138](#)
- exp_E_IntLoop
 - loop_energies.h, [138](#)
- exp_E_Stem
 - loop_energies.h, [137](#)
- expHairpinEnergy
 - part_func.h, [145](#)
- expLoopEnergy
 - part_func.h, [145](#)
- expand_Full
 - RNAstruct.h, [160](#)
- expand_Shapiro
 - RNAstruct.h, [159](#)
- export.ali.bppm
 - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [55](#)
- export.bppm
 - Calculating Partition Functions and Pair Probabilities, [27](#)
- export.co.bppm
 - Partition Function for two hybridized Sequences, [44](#)
- export.cofold_arrays
 - MFE Structures of two hybridized Sequences, [41](#)
- export.cofold_arrays.gq
 - MFE Structures of two hybridized Sequences, [41](#)
- FILENAME_ID_LENGTH
 - utils.h, [170](#)
- FILENAME_MAX_LENGTH
 - utils.h, [169](#)
- FORBIDDEN
 - energy_const.h, [118](#)
- final_cost
 - Searching Sequences for Predefined Structures, [76](#)
- find_saddle
 - findpath.h, [119](#)
- findpath.h
 - find_saddle, [119](#)
 - free_path, [120](#)
 - get_path, [120](#)
- fold
 - Calculating Minimum Free Energy (MFE) Structures, [21](#)
- fold.h

- assign_plist_from_db, [124](#)
- energy_of_circ_struct, [125](#)
- energy_of_move, [123](#)
- energy_of_move_pt, [123](#)
- energy_of_struct, [124](#)
- energy_of_struct_pt, [125](#)
- HairpinE, [124](#)
- initialize_fold, [124](#)
- loop_energy, [123](#)
- LoopEnergy, [124](#)
- parenthesis_structure, [122](#)
- parenthesis_zuker, [122](#)
- fold_par
 - Calculating Minimum Free Energy (MFE) Structures, [21](#)
- fold_vars.h
 - backtrack_type, [130](#)
 - base_pair, [130](#)
 - canonicalBPonly, [131](#)
 - cut_point, [130](#)
 - dangles, [128](#)
 - do_backtrack, [130](#)
 - energy_set, [129](#)
 - iindx, [130](#)
 - james_rule, [129](#)
 - logML, [129](#)
 - noLonelyPairs, [128](#)
 - nonstandards, [129](#)
 - oldAliEn, [129](#)
 - pf_scale, [130](#)
 - pr, [130](#)
 - ribo, [129](#)
 - RibosumFile, [129](#)
 - set_model_details, [128](#)
 - temperature, [129](#)
 - tetra_loop, [129](#)
- folden, [91](#)
- free_path
 - findpath.h, [120](#)
- free_pf_arrays
 - Calculating Partition Functions and Pair Probabilities, [27](#)
- free_profile
 - profiledist.h, [153](#)
- free_sequence_arrays
 - Predicting Consensus Structures from Alignment(s), [50](#)
- free_tree
 - treedist.h, [164](#)
- GASCONST
 - energy_const.h, [117](#)
- get_TwoDfold_variables
 - Calculating MFE representatives of a Distance Based Partitioning, [79](#)
- get_TwoDpfold_variables
 - Calculate Partition Functions of a Distance Based Partitioning, [82](#)
- get_TwoDpfold_variables_from_MFE
 - Calculate Partition Functions of a Distance Based Partitioning, [83](#)
- get_alipf_arrays
 - Predicting Consensus Structures from Alignment(s), [51](#)
- get_boltzmann_factor_copy
 - Change and Precalculate Energy Parameter Sets and Boltzmann Factors, [64](#)
- get_boltzmann_factors
 - Change and Precalculate Energy Parameter Sets and Boltzmann Factors, [63](#)
- get_centroid_struct_pl
 - Compute the centroid structure, [31](#)
- get_centroid_struct_pr
 - Compute the centroid structure, [31](#)
- get_concentrations
 - Partition Function for two hybridized Sequences, [45](#)
- get_gquad_matrix
 - gquad.h, [132](#)
- get_iindx
 - utils.h, [176](#)
- get_indx
 - utils.h, [177](#)
- get_input_line
 - utils.h, [172](#)
- get_line
 - utils.h, [172](#)
- get_monomere_mfes
 - cofold.h, [112](#)
- get_mpi
 - Predicting Consensus Structures from Alignment(s), [49](#)
- get_path
 - findpath.h, [120](#)
- get_pf_arrays
 - Calculating Partition Functions and Pair Probabilities, [28](#)
- get_plist
 - part_func_co.h, [147](#)
- get_scaled_parameters
 - Change and Precalculate Energy Parameter Sets and Boltzmann Factors, [63](#)
- get_scaled_pf_parameters
 - Change and Precalculate Energy Parameter Sets and Boltzmann Factors, [63](#)
- give_up
 - Searching Sequences for Predefined Structures, [76](#)
- gmIRNA
 - PS_dot.h, [155](#)
- gquad.h
 - backtrack_GQuad_IntLoop, [132](#)
 - backtrack_GQuad_IntLoop_L, [132](#)
 - get_gquad_matrix, [132](#)
 - parse_gquad, [132](#)
- HairpinE
 - fold.h, [124](#)
- hamming
 - utils.h, [172](#)

- hamming_bound
 - utils.h, 172
- INF
 - energy_const.h, 117
- INTERVAL, 92
- iindx
 - fold_vars.h, 130
- init_co_pf_fold
 - part_func_co.h, 147
- init_pf_fold
 - part_func.h, 145
- init_pf_foldLP
 - LPfold.h, 140
- initialize_cofold
 - cofold.h, 112
- initialize_fold
 - fold.h, 124
- int_urn
 - utils.h, 171
- interact, 91
- intermediate_t, 92
- inv_verbose
 - Searching Sequences for Predefined Structures, 76
- inverse_fold
 - Searching Sequences for Predefined Structures, 75
- inverse_pf_fold
 - Searching Sequences for Predefined Structures, 76
- james_rule
 - fold_vars.h, 129
- K0
 - energy_const.h, 117
- LIST, 93
- LPfold.h
 - init_pf_foldLP, 140
- LST_BUCKET, 93
- Lfold
 - Local MFE structure Prediction and Z-scores, 58
- Lfoldz
 - Local MFE structure Prediction and Z-scores, 58
- Local MFE consensus structures for Sequence Alignments, 61
 - aliLfold, 61
- Local MFE structure Prediction and Z-scores, 58
 - Lfold, 58
 - Lfoldz, 58
- logML
 - fold_vars.h, 129
- loop_energies.h
 - E_Hairpin, 136
 - E_IntLoop, 135
 - E_Stem, 136
 - exp_E_Hairpin, 138
 - exp_E_IntLoop, 138
 - exp_E_Stem, 137
- loop_energy
 - fold.h, 123
- LoopEnergy
 - fold.h, 124
- MAX2
 - utils.h, 169
- MAX3
 - utils.h, 169
- MAXLOOP
 - energy_const.h, 118
- MEA
 - MEA.h, 141
- MEA.h
 - MEA, 141
- MFE Consensus Structures for Sequence Alignment(s), 52
 - alifold, 52
 - circularfold, 52
- MFE Structures of two hybridized Sequences, 40
 - cofold, 40
 - export_cofold_arrays, 41
 - export_cofold_arrays_gq, 41
- MIN2
 - utils.h, 169
- MIN3
 - utils.h, 169
- Make_bp_profile
 - profiledist.h, 153
- Make_bp_profile_bppm
 - profiledist.h, 153
- make_loop_index_pt
 - utils.h, 175
- make_pair_table
 - utils.h, 174
- make_pair_table_snoop
 - utils.h, 175
- Make_swString
 - stringdist.h, 161
- make_tree
 - treedist.h, 164
- mean_bp_dist
 - part_func.h, 145
- mean_bp_distance
 - Calculating Partition Functions and Pair Probabilities, 28
- mean_bp_distance_pr
 - Calculating Partition Functions and Pair Probabilities, 29
- model_detailsT, 93
 - dangles, 94
- move_t, 94
- NBPAIRS
 - energy_const.h, 118
- nc_fact
 - Predicting Consensus Structures from Alignment(s), 51
- noLonelyPairs
 - fold_vars.h, 128

- nonstandards
 - fold_vars.h, 129
- nrrerror
 - utils.h, 170
- oldAliEn
 - fold_vars.h, 129
- PAIR, 94
- PS_dot.h
 - aliPS_color_aln, 157
 - gmlRNA, 155
 - PS_dot_plot, 157
 - PS_dot_plot_list, 156
 - PS_rna_plot, 155
 - PS_rna_plot_a, 155
 - ssv_rna_plot, 156
 - svg_rna_plot, 156
 - xrna_plot, 156
- PS_dot_plot
 - PS_dot.h, 157
- PS_dot_plot_list
 - PS_dot.h, 156
- PS_rna_plot
 - PS_dot.h, 155
- PS_rna_plot_a
 - PS_dot.h, 155
- pack_structure
 - utils.h, 174
- pair_info, 95
- pairpro, 96
- paramT, 96
- parenthesis_structure
 - fold.h, 122
- parenthesis_zuker
 - fold.h, 122
- parse_gquad
 - gquad.h, 132
- parse_structure
 - RNAstruct.h, 161
- Parsing and Comparing - Functions to Manipulate Structures, 88
- part_func.h
 - centroid, 145
 - expHairpinEnergy, 145
 - expLoopEnergy, 145
 - init_pf_fold, 145
 - mean_bp_dist, 145
- part_func_co.h
 - get_plist, 147
 - init_co_pf_fold, 147
- Partition Function and Base Pair Probabilities for Sequence Alignment(s), 54
 - alipf_circ_fold, 55
 - alipf_fold, 54
 - alipf_fold_par, 54
 - export_ali_bppm, 55
- Partition Function for two hybridized Sequences, 42
 - co_pf_fold, 43
 - co_pf_fold_par, 43
 - compute_probabilities, 45
 - export_co_bppm, 44
 - get_concentrations, 45
 - update_co_pf_params, 44
 - update_co_pf_params_par, 44
- Partition Function for two hybridized Sequences as a stepwise Process, 46
 - pf_interact, 47
 - pf_unstru, 46
- Partition functions for locally stable secondary structures, 59
 - pfl_fold, 59
 - putoutpU_prob, 60
 - putoutpU_prob_bin, 60
 - update_pf_paramsLP, 59
- path_t, 97
- pbacktrack
 - Stochastic backtracking in the Ensemble, 37
- pbacktrack_circ
 - Stochastic backtracking in the Ensemble, 38
- pf_circ_fold
 - Calculating Partition Functions and Pair Probabilities, 26
- pf_fold
 - Calculating Partition Functions and Pair Probabilities, 25
- pf_fold_par
 - Calculating Partition Functions and Pair Probabilities, 24
- pf_interact
 - Partition Function for two hybridized Sequences as a stepwise Process, 47
- pf_paramT, 97
 - alpha, 98
- pf_scale
 - fold_vars.h, 130
- pf_unstru
 - Partition Function for two hybridized Sequences as a stepwise Process, 46
- pfl_fold
 - Partition functions for locally stable secondary structures, 59
- plist, 98
- plot_layouts.h
 - rna_plot_type, 151
 - simple_circplot_coordinates, 151
 - simple_xy_coordinates, 151
- Postorder_list, 98
- pr
 - fold_vars.h, 130
- Predicting Consensus Structures from Alignment(s), 48
 - alloc_sequence_arrays, 50
 - cv_fact, 51
 - encode_ali_sequence, 49
 - energy_of_alistruct, 49
 - free_sequence_arrays, 50
 - get_alipf_arrays, 51

- get_mpi, [49](#)
- nc_fact, [51](#)
- Predicting Locally stable structures of large sequences, [57](#)
- print_tty_constraint
 - utils.h, [176](#)
- print_tty_input_seq
 - utils.h, [175](#)
- print_tty_input_seq_str
 - utils.h, [175](#)
- profile_edit_distance
 - profiledist.h, [153](#)
- profiledist.h
 - free_profile, [153](#)
 - Make_bp_profile, [153](#)
 - Make_bp_profile_bppm, [153](#)
 - profile_edit_distance, [153](#)
- pu_contrib, [98](#)
- pu_out, [99](#)
- putoutpU_prob
 - Partition functions for locally stable secondary structures, [60](#)
- putoutpU_prob_bin
 - Partition functions for locally stable secondary structures, [60](#)
- RNA Secondary Structure Folding, [17](#)
- RNAstruct.h
 - add_root, [159](#)
 - b2C, [159](#)
 - b2HIT, [159](#)
 - b2Shapiro, [159](#)
 - expand_Full, [160](#)
 - expand_Shapiro, [159](#)
 - parse_structure, [161](#)
 - unexpand_Full, [160](#)
 - unexpand_aligned_F, [160](#)
 - unweight, [160](#)
- random_string
 - utils.h, [171](#)
- read_parameter_file
 - Reading/Writing energy parameter sets from/to File, [65](#)
- read_record
 - utils.h, [173](#)
- Reading/Writing energy parameter sets from/to File, [65](#)
 - read_parameter_file, [65](#)
 - write_parameter_file, [65](#)
- ribo
 - fold_vars.h, [129](#)
- RibosumFile
 - fold_vars.h, [129](#)
- rna_plot_type
 - plot_layouts.h, [151](#)
- SOLUTION, [100](#)
- STR
 - utils.h, [169](#)
- scale_parameters
 - Change and Precalculate Energy Parameter Sets and Boltzmann Factors, [63](#)
 - Searching Sequences for Predefined Structures, [75](#)
 - final_cost, [76](#)
 - give_up, [76](#)
 - inv_verbose, [76](#)
 - inverse_fold, [75](#)
 - inverse_pf_fold, [76](#)
 - sect, [99](#)
 - set_model_details
 - fold_vars.h, [128](#)
 - simple_circplot_coordinates
 - plot_layouts.h, [151](#)
 - simple_xy_coordinates
 - plot_layouts.h, [151](#)
 - snoopT, [99](#)
 - space
 - utils.h, [170](#)
 - ssv_rna_plot
 - PS_dot.h, [156](#)
 - st_back
 - Stochastic backtracking in the Ensemble, [38](#)
 - Stochastic backtracking in the Ensemble, [37](#)
 - pbacktrack, [37](#)
 - pbacktrack_circ, [38](#)
 - st_back, [38](#)
 - Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), [56](#)
 - alipbacktrack, [56](#)
 - Stochastic Backtracking of Structures from Distance Based Partitioning, [85](#)
 - TwoDpfold_pbacktrack, [85](#)
 - TwoDpfold_pbacktrack5, [86](#)
 - str_DNA2RNA
 - utils.h, [176](#)
 - str_uppercase
 - utils.h, [176](#)
 - string_edit_distance
 - stringdist.h, [162](#)
 - stringdist.h
 - Make_swString, [161](#)
 - string_edit_distance, [162](#)
 - struct_en, [100](#)
 - subopt
 - Suboptimal structures within an energy band around the MFE, [35](#)
 - subopt_circ
 - Suboptimal structures within an energy band around the MFE, [36](#)
 - Suboptimal structures according to Zuker et al. 1989, [34](#)
 - zukersubopt, [34](#)
 - Suboptimal structures within an energy band around the MFE, [35](#)
 - subopt, [35](#)
 - subopt_circ, [36](#)
 - svg_rna_plot
 - PS_dot.h, [156](#)
 - svm_model, [100](#)

- swString, [100](#)
- TURN
 - energy_const.h, [118](#)
- temperature
 - fold_vars.h, [129](#)
- tetra_loop
 - fold_vars.h, [129](#)
- time_stamp
 - utils.h, [171](#)
- Tree, [101](#)
- tree_edit_distance
 - treedist.h, [164](#)
- treedist.h
 - free_tree, [164](#)
 - make_tree, [164](#)
 - tree_edit_distance, [164](#)
- TwoDfold_backtrack_f5
 - Calculating MFE representatives of a Distance Based Partitioning, [81](#)
- TwoDfold_solution, [101](#)
- TwoDfold_vars, [102](#)
- TwoDfoldList
 - Calculating MFE representatives of a Distance Based Partitioning, [80](#)
- TwoDpfold_pbacktrack
 - Stochastic Backtracking of Structures from Distance Based Partitioning, [85](#)
- TwoDpfold_pbacktrack5
 - Stochastic Backtracking of Structures from Distance Based Partitioning, [86](#)
- TwoDpfold_solution, [103](#)
- TwoDpfold_vars, [103](#)
- TwoDpfoldList
 - Calculate Partition Functions of a Distance Based Partitioning, [83](#)
- unexpand_Full
 - RNAstruct.h, [160](#)
- unexpand_aligned_F
 - RNAstruct.h, [160](#)
- unpack_structure
 - utils.h, [174](#)
- unweight
 - RNAstruct.h, [160](#)
- update_alifold_params
 - alifold.h, [110](#)
- update_co_pf_params
 - Partition Function for two hybridized Sequences, [44](#)
- update_co_pf_params_par
 - Partition Function for two hybridized Sequences, [44](#)
- update_pf_params
 - Calculating Partition Functions and Pair Probabilities, [27](#)
- update_pf_paramsLP
 - Partition functions for locally stable secondary structures, [59](#)
- urn
 - utils.h, [171](#)
- utils.h
 - alimake_pair_table, [175](#)
 - constrain_ptypes, [177](#)
 - copy_pair_table, [175](#)
 - FILENAME_ID_LENGTH, [170](#)
 - FILENAME_MAX_LENGTH, [169](#)
 - get_iindx, [176](#)
 - get_indx, [177](#)
 - get_input_line, [172](#)
 - get_line, [172](#)
 - hamming, [172](#)
 - hamming_bound, [172](#)
 - int_urn, [171](#)
 - MAX2, [169](#)
 - MAX3, [169](#)
 - MIN2, [169](#)
 - MIN3, [169](#)
 - make_loop_index_pt, [175](#)
 - make_pair_table, [174](#)
 - make_pair_table_snoop, [175](#)
 - nrerror, [170](#)
 - pack_structure, [174](#)
 - print_tty_constraint, [176](#)
 - print_tty_input_seq, [175](#)
 - print_tty_input_seq_str, [175](#)
 - random_string, [171](#)
 - read_record, [173](#)
 - STR, [169](#)
 - space, [170](#)
 - str_DNA2RNA, [176](#)
 - str_uppercase, [176](#)
 - time_stamp, [171](#)
 - unpack_structure, [174](#)
 - urn, [171](#)
 - VRNA_CONSTRAINT_ALL, [169](#)
 - VRNA_CONSTRAINT_DOT, [168](#)
 - VRNA_CONSTRAINT_G, [169](#)
 - VRNA_CONSTRAINT_X, [168](#)
 - VRNA_INPUT_COMMENT, [168](#)
 - VRNA_INPUT_ERROR, [167](#)
 - VRNA_INPUT_MISC, [167](#)
 - VRNA_INPUT_NO_REST, [168](#)
 - VRNA_INPUT_NO_SPAN, [168](#)
 - VRNA_INPUT_QUIT, [167](#)
 - VRNA_INPUT_SEQUENCE, [167](#)
 - warn_user, [170](#)
 - XSTR, [169](#)
 - xrealloc, [170](#)
 - xsubi, [178](#)
- VRNA_CONSTRAINT_ALL
 - utils.h, [169](#)
- VRNA_CONSTRAINT_DOT
 - utils.h, [168](#)
- VRNA_CONSTRAINT_G
 - utils.h, [169](#)
- VRNA_CONSTRAINT_PIPE
 - utils.h, [168](#)
- VRNA_CONSTRAINT_X

- utils.h, [168](#)
- VRNA_INPUT_COMMENT
 - utils.h, [168](#)
- VRNA_INPUT_ERROR
 - utils.h, [167](#)
- VRNA_INPUT_MISC
 - utils.h, [167](#)
- VRNA_INPUT_NO_REST
 - utils.h, [168](#)
- VRNA_INPUT_NO_SPAN
 - utils.h, [168](#)
- VRNA_INPUT_QUIT
 - utils.h, [167](#)
- VRNA_INPUT_SEQUENCE
 - utils.h, [167](#)

- warn_user
 - utils.h, [170](#)
- write_parameter_file
 - Reading/Writing energy parameter sets from/to File, [65](#)

- XSTR
 - utils.h, [169](#)
- xrealloc
 - utils.h, [170](#)
- xrna_plot
 - PS_dot.h, [156](#)
- xsubi
 - utils.h, [178](#)

- zukersubopt
 - Suboptimal structures according to Zuker et al. 1989, [34](#)