

HNCO

0.12

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Namespace List . . . . .	1
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>7</b>
3.1	Class List . . . . .	7
<b>4</b>	<b>Namespace Documentation</b>	<b>13</b>
4.1	hnco Namespace Reference . . . . .	13
4.1.1	Detailed Description . . . . .	17
4.1.2	Typedef Documentation . . . . .	17
4.1.2.1	bit_t . . . . .	17
4.1.2.2	sparse_bit_matrix_t . . . . .	17
4.1.2.3	sparse_bit_vector_t . . . . .	17
4.1.3	Function Documentation . . . . .	17
4.1.3.1	bm_add_rows() . . . . .	18
4.1.3.2	bm_identity() . . . . .	18
4.1.3.3	bm_invert() . . . . .	18
4.1.3.4	bm_multiply() . . . . .	19
4.1.3.5	bm_rank() . . . . .	19
4.1.3.6	bm_row_echelon_form() . . . . .	19
4.1.3.7	bm_solve() . . . . .	19
4.1.3.8	bm_solve_upper_triangular() . . . . .	20

4.1.3.9	<a href="#">bv_from_vector_bool()</a>	21
4.1.3.10	<a href="#">bv_to_vector_bool()</a>	21
4.1.3.11	<a href="#">perm_identity()</a>	21
4.1.3.12	<a href="#">perm_random()</a>	22
4.1.3.13	<a href="#">sbm_multiply()</a>	22
4.2	<a href="#">hnco::algorithm Namespace Reference</a>	22
4.2.1	<a href="#">Detailed Description</a>	25
4.3	<a href="#">hnco::algorithm::bm_pbil Namespace Reference</a>	25
4.3.1	<a href="#">Detailed Description</a>	25
4.4	<a href="#">hnco::algorithm::eda Namespace Reference</a>	25
4.4.1	<a href="#">Detailed Description</a>	25
4.5	<a href="#">hnco::algorithm::hea Namespace Reference</a>	26
4.5.1	<a href="#">Detailed Description</a>	26
4.6	<a href="#">hnco::exception Namespace Reference</a>	26
4.6.1	<a href="#">Detailed Description</a>	26
4.7	<a href="#">hnco::function Namespace Reference</a>	27
4.7.1	<a href="#">Detailed Description</a>	29
4.8	<a href="#">hnco::neighborhood Namespace Reference</a>	29
4.8.1	<a href="#">Detailed Description</a>	29
4.9	<a href="#">hnco::random Namespace Reference</a>	29
4.9.1	<a href="#">Detailed Description</a>	29

<b>5</b>	<b>Class Documentation</b>	<b>31</b>
5.1	AbstractLabs Class Reference	31
5.1.1	Detailed Description	32
5.2	AbstractMaxSat Class Reference	32
5.2.1	Detailed Description	33
5.2.2	Member Function Documentation	33
5.2.2.1	load()	33
5.2.3	Member Data Documentation	33
5.2.3.1	_expression	33
5.3	AdditiveGaussianNoise Class Reference	34
5.3.1	Detailed Description	35
5.3.2	Member Function Documentation	35
5.3.2.1	get_maximum()	35
5.3.2.2	has_known_maximum()	35
5.4	AffineMap Class Reference	36
5.4.1	Detailed Description	37
5.4.2	Member Function Documentation	37
5.4.2.1	is_surjective()	37
5.4.2.2	random()	37
5.5	Algorithm Class Reference	38
5.5.1	Detailed Description	40
5.5.2	Member Function Documentation	40
5.5.2.1	set_solution()	40
5.5.2.2	update_solution()	41
5.5.3	Member Data Documentation	41
5.5.3.1	_functions	41
5.6	BernoulliProcess Class Reference	41
5.6.1	Detailed Description	42
5.6.2	Constructor & Destructor Documentation	42
5.6.2.1	BernoulliProcess() [1/2]	42

5.6.2.2	<a href="#">BernoulliProcess()</a> [2/2]	43
5.6.3	<a href="#">Member Function Documentation</a>	43
5.6.3.1	<a href="#">set_allow_stay()</a>	43
5.6.3.2	<a href="#">set_probability()</a>	43
5.7	<a href="#">BiasedCrossover Class Reference</a>	44
5.7.1	<a href="#">Detailed Description</a>	44
5.7.2	<a href="#">Member Function Documentation</a>	44
5.7.2.1	<a href="#">breed()</a>	44
5.8	<a href="#">BitHerding Class Reference</a>	45
5.8.1	<a href="#">Detailed Description</a>	46
5.8.2	<a href="#">Member Enumeration Documentation</a>	46
5.8.2.1	<a href="#">anonymous enum</a>	46
5.9	<a href="#">BitMoment Struct Reference</a>	47
5.9.1	<a href="#">Detailed Description</a>	48
5.10	<a href="#">BmPbil Class Reference</a>	48
5.10.1	<a href="#">Detailed Description</a>	50
5.10.2	<a href="#">Member Enumeration Documentation</a>	50
5.10.2.1	<a href="#">anonymous enum</a>	50
5.10.2.2	<a href="#">anonymous enum</a>	51
5.10.2.3	<a href="#">anonymous enum</a>	51
5.10.3	<a href="#">Member Function Documentation</a>	51
5.10.3.1	<a href="#">set_selection_size()</a>	51
5.11	<a href="#">Cache Class Reference</a>	52
5.11.1	<a href="#">Detailed Description</a>	53
5.11.2	<a href="#">Constructor &amp; Destructor Documentation</a>	53
5.11.2.1	<a href="#">Cache()</a>	53
5.11.3	<a href="#">Member Function Documentation</a>	53
5.11.3.1	<a href="#">provides_incremental_evaluation()</a>	54
5.12	<a href="#">CallCounter Class Reference</a>	54
5.12.1	<a href="#">Detailed Description</a>	55

5.13 CompactGa Class Reference . . . . .	55
5.13.1 Detailed Description . . . . .	57
5.14 CompleteSearch Class Reference . . . . .	57
5.14.1 Detailed Description . . . . .	58
5.15 Crossover Class Reference . . . . .	58
5.15.1 Detailed Description . . . . .	58
5.15.2 Member Function Documentation . . . . .	59
5.15.2.1 breed() . . . . .	59
5.16 DeceptiveJump Class Reference . . . . .	59
5.16.1 Detailed Description . . . . .	60
5.16.2 Member Function Documentation . . . . .	60
5.16.2.1 get_maximum() . . . . .	60
5.16.2.2 has_known_maximum() . . . . .	61
5.17 EqualProducts Class Reference . . . . .	61
5.17.1 Detailed Description . . . . .	62
5.17.2 Member Function Documentation . . . . .	62
5.17.2.1 random() . . . . .	62
5.18 Error Class Reference . . . . .	63
5.18.1 Detailed Description . . . . .	64
5.19 ProgressTracker::Event Struct Reference . . . . .	64
5.19.1 Detailed Description . . . . .	64
5.20 Exception Class Reference . . . . .	64
5.20.1 Detailed Description . . . . .	65
5.21 Factorization Class Reference . . . . .	65
5.21.1 Detailed Description . . . . .	66
5.21.2 Constructor & Destructor Documentation . . . . .	66
5.21.2.1 Factorization() . . . . .	66
5.22 FirstAscentHillClimbing Class Reference . . . . .	67
5.22.1 Detailed Description . . . . .	68
5.23 FourPeaks Class Reference . . . . .	68

5.23.1 Detailed Description . . . . .	69
5.23.2 Member Function Documentation . . . . .	70
5.23.2.1 get_maximum() . . . . .	70
5.23.2.2 has_known_maximum() . . . . .	70
5.24 Function Class Reference . . . . .	71
5.24.1 Detailed Description . . . . .	72
5.24.2 Member Function Documentation . . . . .	72
5.24.2.1 compute_walsh_transform() . . . . .	72
5.24.2.2 get_maximum() . . . . .	73
5.24.2.3 incremental_eval() . . . . .	73
5.24.2.4 provides_incremental_evaluation() . . . . .	74
5.24.2.5 safe_eval() . . . . .	74
5.25 FunctionController Class Reference . . . . .	75
5.25.1 Detailed Description . . . . .	76
5.25.2 Member Function Documentation . . . . .	76
5.25.2.1 provides_incremental_evaluation() . . . . .	76
5.26 FunctionDecorator Class Reference . . . . .	76
5.26.1 Detailed Description . . . . .	77
5.27 FunctionMapComposition Class Reference . . . . .	77
5.27.1 Detailed Description . . . . .	78
5.27.2 Constructor & Destructor Documentation . . . . .	78
5.27.2.1 FunctionMapComposition() . . . . .	78
5.27.3 Member Function Documentation . . . . .	79
5.27.3.1 get_maximum() . . . . .	79
5.27.3.2 has_known_maximum() . . . . .	79
5.28 FunctionModifier Class Reference . . . . .	80
5.28.1 Detailed Description . . . . .	80
5.29 FunctionPlugin Class Reference . . . . .	81
5.29.1 Detailed Description . . . . .	82
5.29.2 Constructor & Destructor Documentation . . . . .	82



5.29.2.1	FunctionPlugin()	82
5.30	GeneticAlgorithm Class Reference	82
5.30.1	Detailed Description	84
5.30.2	Constructor & Destructor Documentation	84
5.30.2.1	GeneticAlgorithm()	84
5.30.3	Member Function Documentation	85
5.30.3.1	set_allow_stay()	85
5.31	HammingBall Class Reference	85
5.31.1	Detailed Description	86
5.31.2	Constructor & Destructor Documentation	86
5.31.2.1	HammingBall()	86
5.32	HammingSphere Class Reference	87
5.32.1	Detailed Description	88
5.32.2	Constructor & Destructor Documentation	88
5.32.2.1	HammingSphere()	88
5.33	HammingSphereIterator Class Reference	88
5.33.1	Detailed Description	89
5.33.2	Constructor & Destructor Documentation	90
5.33.2.1	HammingSphereIterator()	90
5.34	Hboa Class Reference	90
5.34.1	Detailed Description	91
5.35	Hea< Moment, Herding > Class Template Reference	91
5.35.1	Detailed Description	94
5.35.2	Member Enumeration Documentation	94
5.35.2.1	anonymous enum	94
5.35.3	Constructor & Destructor Documentation	94
5.35.3.1	Hea()	94
5.35.4	Member Function Documentation	95
5.35.4.1	set_reset_period()	95
5.35.4.2	set_selection_size()	95

5.36 Hiff Class Reference . . . . .	96
5.36.1 Detailed Description . . . . .	96
5.36.2 Member Function Documentation . . . . .	97
5.36.2.1 get_maximum() . . . . .	97
5.36.2.2 has_known_maximum() . . . . .	97
5.37 HncoEvaluator Class Reference . . . . .	97
5.37.1 Detailed Description . . . . .	98
5.38 Hypercubeliterator Class Reference . . . . .	98
5.38.1 Detailed Description . . . . .	99
5.39 Injection Class Reference . . . . .	99
5.39.1 Detailed Description . . . . .	100
5.39.2 Constructor & Destructor Documentation . . . . .	100
5.39.2.1 Injection() . . . . .	100
5.40 IterativeAlgorithm Class Reference . . . . .	101
5.40.1 Detailed Description . . . . .	102
5.40.2 Constructor & Destructor Documentation . . . . .	102
5.40.2.1 IterativeAlgorithm() . . . . .	102
5.40.3 Member Function Documentation . . . . .	103
5.40.3.1 maximize() . . . . .	103
5.40.3.2 set_num_iterations() . . . . .	103
5.41 Iterator Class Reference . . . . .	104
5.41.1 Detailed Description . . . . .	105
5.42 Jump Class Reference . . . . .	105
5.42.1 Detailed Description . . . . .	106
5.42.2 Member Function Documentation . . . . .	106
5.42.2.1 get_maximum() . . . . .	106
5.42.2.2 has_known_maximum() . . . . .	106
5.43 Labs Class Reference . . . . .	107
5.43.1 Detailed Description . . . . .	107
5.44 LabsMeritFactor Class Reference . . . . .	108

5.44.1 Detailed Description . . . . .	108
5.45 LastEvaluation Class Reference . . . . .	109
5.45.1 Detailed Description . . . . .	109
5.46 LeadingOnes Class Reference . . . . .	109
5.46.1 Detailed Description . . . . .	110
5.46.2 Member Function Documentation . . . . .	110
5.46.2.1 get_maximum() . . . . .	110
5.46.2.2 has_known_maximum() . . . . .	111
5.47 LinearFunction Class Reference . . . . .	111
5.47.1 Detailed Description . . . . .	112
5.47.2 Member Function Documentation . . . . .	112
5.47.2.1 has_known_maximum() . . . . .	112
5.47.2.2 random() . . . . .	112
5.48 LinearMap Class Reference . . . . .	113
5.48.1 Detailed Description . . . . .	114
5.48.2 Member Function Documentation . . . . .	114
5.48.2.1 is_surjective() . . . . .	114
5.48.2.2 random() . . . . .	114
5.49 LocalMaximum Class Reference . . . . .	115
5.49.1 Detailed Description . . . . .	116
5.50 LogContext Class Reference . . . . .	116
5.50.1 Detailed Description . . . . .	116
5.51 LongPath Class Reference . . . . .	117
5.51.1 Detailed Description . . . . .	118
5.51.2 Member Function Documentation . . . . .	118
5.51.2.1 get_maximum() . . . . .	118
5.51.2.2 has_known_maximum() . . . . .	119
5.52 Ltga Class Reference . . . . .	119
5.52.1 Detailed Description . . . . .	120
5.53 Map Class Reference . . . . .	120

5.53.1 Detailed Description . . . . .	121
5.53.2 Member Function Documentation . . . . .	121
5.53.2.1 is_surjective() . . . . .	121
5.54 MapComposition Class Reference . . . . .	122
5.54.1 Detailed Description . . . . .	122
5.54.2 Constructor & Destructor Documentation . . . . .	123
5.54.2.1 MapComposition() . . . . .	123
5.54.3 Member Function Documentation . . . . .	123
5.54.3.1 is_surjective() . . . . .	123
5.55 MaximumReached Class Reference . . . . .	124
5.55.1 Detailed Description . . . . .	124
5.56 MaxNae3Sat Class Reference . . . . .	125
5.56.1 Detailed Description . . . . .	125
5.56.2 Member Function Documentation . . . . .	125
5.56.2.1 load() . . . . .	125
5.57 MaxSat Class Reference . . . . .	126
5.57.1 Detailed Description . . . . .	127
5.57.2 Member Function Documentation . . . . .	127
5.57.2.1 random() [1/2] . . . . .	127
5.57.2.2 random() [2/2] . . . . .	127
5.58 Mimic Class Reference . . . . .	128
5.58.1 Detailed Description . . . . .	129
5.59 Mmas Class Reference . . . . .	130
5.59.1 Detailed Description . . . . .	131
5.60 Model Class Reference . . . . .	131
5.60.1 Detailed Description . . . . .	132
5.61 ModelParameters Class Reference . . . . .	132
5.61.1 Detailed Description . . . . .	133
5.62 MuCommaLambdaEa Class Reference . . . . .	133
5.62.1 Detailed Description . . . . .	134

5.62.2	Constructor & Destructor Documentation	134
5.62.2.1	MuCommaLambdaEa()	134
5.62.3	Member Function Documentation	135
5.62.3.1	set_allow_stay()	135
5.63	MultiBitFlip Class Reference	135
5.63.1	Detailed Description	136
5.63.2	Constructor & Destructor Documentation	136
5.63.2.1	MultiBitFlip()	136
5.63.3	Member Function Documentation	136
5.63.3.1	bernoulli_trials()	136
5.63.3.2	reservoir_sampling()	137
5.64	MuPlusLambdaEa Class Reference	137
5.64.1	Detailed Description	138
5.64.2	Constructor & Destructor Documentation	139
5.64.2.1	MuPlusLambdaEa()	139
5.64.3	Member Function Documentation	139
5.64.3.1	set_allow_stay()	139
5.65	Needle Class Reference	140
5.65.1	Detailed Description	140
5.65.2	Member Function Documentation	141
5.65.2.1	get_maximum()	141
5.65.2.2	has_known_maximum()	141
5.66	Negation Class Reference	142
5.66.1	Detailed Description	143
5.66.2	Member Function Documentation	143
5.66.2.1	get_maximum()	143
5.66.2.2	has_known_maximum()	143
5.66.2.3	provides_incremental_evaluation()	144
5.67	Neighborhood Class Reference	144
5.67.1	Detailed Description	146

5.67.2	Constructor & Destructor Documentation	146
5.67.2.1	Neighborhood()	146
5.67.3	Member Function Documentation	146
5.67.3.1	map()	146
5.67.3.2	mutate()	147
5.68	NeighborhoodIterator Class Reference	147
5.68.1	Detailed Description	148
5.68.2	Constructor & Destructor Documentation	148
5.68.2.1	NeighborhoodIterator()	148
5.69	NkLandscape Class Reference	148
5.69.1	Detailed Description	149
5.69.2	Member Function Documentation	150
5.69.2.1	random()	150
5.70	NpsPbil Class Reference	150
5.70.1	Detailed Description	152
5.71	OnBudgetFunction Class Reference	152
5.71.1	Detailed Description	154
5.71.2	Member Function Documentation	154
5.71.2.1	eval()	154
5.71.2.2	incremental_eval()	154
5.71.2.3	update()	155
5.72	OneMax Class Reference	155
5.72.1	Detailed Description	156
5.72.2	Member Function Documentation	156
5.72.2.1	get_maximum()	157
5.72.2.2	has_known_maximum()	157
5.72.2.3	provides_incremental_evaluation()	157
5.73	OnePlusLambdaCommaLambdaGa Class Reference	158
5.73.1	Detailed Description	159
5.73.2	Constructor & Destructor Documentation	159

5.73.2.1	OnePlusLambdaCommaLambdaGa()	159
5.74	OnePlusOneEa Class Reference	160
5.74.1	Detailed Description	161
5.74.2	Constructor & Destructor Documentation	161
5.74.2.1	OnePlusOneEa()	161
5.74.3	Member Function Documentation	162
5.74.3.1	set_allow_stay()	162
5.74.3.2	set_num_iterations()	162
5.75	ParameterLessPopulationPyramid Class Reference	162
5.75.1	Detailed Description	163
5.76	Pbil Class Reference	164
5.76.1	Detailed Description	165
5.77	Permutation Class Reference	165
5.77.1	Detailed Description	166
5.77.2	Member Function Documentation	166
5.77.2.1	is_surjective()	167
5.78	Plateau Class Reference	167
5.78.1	Detailed Description	168
5.78.2	Member Function Documentation	168
5.78.2.1	get_maximum()	168
5.78.2.2	has_known_maximum()	168
5.79	PointValueException Class Reference	169
5.79.1	Detailed Description	169
5.80	Population Class Reference	170
5.80.1	Detailed Description	171
5.80.2	Member Function Documentation	172
5.80.2.1	comma_selection() [1/2]	172
5.80.2.2	comma_selection() [2/2]	172
5.80.2.3	get_best_bv() [1/4]	173
5.80.2.4	get_best_bv() [2/4]	173

5.80.2.5	<a href="#">get_best_bv()</a> [3/4]	173
5.80.2.6	<a href="#">get_best_bv()</a> [4/4]	173
5.80.2.7	<a href="#">get_best_value()</a> [1/2]	174
5.80.2.8	<a href="#">get_best_value()</a> [2/2]	174
5.80.2.9	<a href="#">get_worst_bv()</a> [1/2]	174
5.80.2.10	<a href="#">get_worst_bv()</a> [2/2]	175
5.80.2.11	<a href="#">plus_selection()</a> [1/2]	175
5.80.2.12	<a href="#">plus_selection()</a> [2/2]	176
5.80.3	Member Data Documentation	176
5.80.3.1	<a href="#">_compare_index_value</a>	176
5.80.3.2	<a href="#">_lookup</a>	176
5.81	PriorNoise Class Reference	177
5.81.1	Detailed Description	178
5.81.2	Member Function Documentation	178
5.81.2.1	<a href="#">get_maximum()</a>	178
5.81.2.2	<a href="#">has_known_maximum()</a>	178
5.81.2.3	<a href="#">provides_incremental_evaluation()</a>	179
5.82	ProgressTracker Class Reference	179
5.82.1	Detailed Description	181
5.82.2	Member Function Documentation	181
5.82.2.1	<a href="#">eval()</a>	181
5.82.2.2	<a href="#">get_last_improvement()</a>	181
5.82.2.3	<a href="#">incremental_eval()</a>	181
5.82.2.4	<a href="#">update()</a>	182
5.83	ProgressTrackerContext Class Reference	182
5.83.1	Detailed Description	183
5.84	Projection Class Reference	183
5.84.1	Detailed Description	184
5.84.2	Constructor & Destructor Documentation	184
5.84.2.1	<a href="#">Projection()</a>	184



5.84.3	Member Function Documentation	185
5.84.3.1	is_surjective()	185
5.85	PvAlgorithm Class Reference	185
5.85.1	Detailed Description	187
5.85.2	Member Enumeration Documentation	187
5.85.2.1	anonymous enum	187
5.86	Qubo Class Reference	187
5.86.1	Detailed Description	188
5.86.2	Member Function Documentation	189
5.86.2.1	load()	189
5.86.3	Member Data Documentation	189
5.86.3.1	_q	189
5.87	Random Struct Reference	190
5.87.1	Detailed Description	190
5.88	RandomLocalSearch Class Reference	190
5.88.1	Detailed Description	191
5.88.2	Member Function Documentation	192
5.88.2.1	set_patience()	192
5.89	RandomSearch Class Reference	192
5.89.1	Detailed Description	193
5.90	RandomWalk Class Reference	193
5.90.1	Detailed Description	194
5.91	Restart Class Reference	195
5.91.1	Detailed Description	196
5.92	Ridge Class Reference	196
5.92.1	Detailed Description	197
5.92.2	Member Function Documentation	197
5.92.2.1	get_maximum()	197
5.92.2.2	has_known_maximum()	197
5.93	SimulatedAnnealing Class Reference	198

5.93.1 Detailed Description . . . . .	199
5.93.2 Member Function Documentation . . . . .	199
5.93.2.1 init_beta() . . . . .	199
5.94 SingleBitFlip Class Reference . . . . .	200
5.94.1 Detailed Description . . . . .	200
5.95 SingleBitFlipIteator Class Reference . . . . .	201
5.95.1 Detailed Description . . . . .	201
5.95.2 Constructor & Destructor Documentation . . . . .	202
5.95.2.1 SingleBitFlipIteator() . . . . .	202
5.96 SinusSummationCancellation Class Reference . . . . .	202
5.96.1 Detailed Description . . . . .	203
5.97 SixPeaks Class Reference . . . . .	203
5.97.1 Detailed Description . . . . .	204
5.97.2 Member Function Documentation . . . . .	205
5.97.2.1 get_maximum() . . . . .	205
5.97.2.2 has_known_maximum() . . . . .	205
5.98 SpinHerdng Class Reference . . . . .	205
5.98.1 Detailed Description . . . . .	207
5.98.2 Member Enumeration Documentation . . . . .	207
5.98.2.1 anonymous enum . . . . .	207
5.98.3 Constructor & Destructor Documentation . . . . .	208
5.98.3.1 SpinHerdng() . . . . .	208
5.98.4 Member Function Documentation . . . . .	208
5.98.4.1 q_variation() . . . . .	208
5.99 SpinMoment Struct Reference . . . . .	208
5.99.1 Detailed Description . . . . .	209
5.99.2 Member Data Documentation . . . . .	209
5.99.2.1 _second . . . . .	210
5.100SteepestAscentHillClimbing Class Reference . . . . .	210
5.100.1 Detailed Description . . . . .	211

5.101StopOnMaximum Class Reference . . . . .	211
5.101.1 Detailed Description . . . . .	212
5.101.2 Constructor & Destructor Documentation . . . . .	212
5.101.2.1 StopOnMaximum() . . . . .	212
5.101.3 Member Function Documentation . . . . .	213
5.101.3.1 eval() . . . . .	213
5.101.3.2 incremental_eval() . . . . .	213
5.101.3.3 update() . . . . .	213
5.102StopOnTarget Class Reference . . . . .	214
5.102.1 Detailed Description . . . . .	215
5.102.2 Constructor & Destructor Documentation . . . . .	215
5.102.2.1 StopOnTarget() . . . . .	215
5.102.3 Member Function Documentation . . . . .	216
5.102.3.1 eval() . . . . .	216
5.102.3.2 incremental_eval() . . . . .	216
5.102.3.3 update() . . . . .	217
5.103StopWatch Class Reference . . . . .	217
5.103.1 Detailed Description . . . . .	218
5.104SummationCancellation Class Reference . . . . .	218
5.104.1 Detailed Description . . . . .	219
5.104.2 Constructor & Destructor Documentation . . . . .	219
5.104.2.1 SummationCancellation() . . . . .	219
5.104.3 Member Function Documentation . . . . .	220
5.104.3.1 has_known_maximum() . . . . .	220
5.105TargetReached Class Reference . . . . .	220
5.105.1 Detailed Description . . . . .	221
5.106TournamentSelection Class Reference . . . . .	221
5.106.1 Detailed Description . . . . .	222
5.106.2 Member Function Documentation . . . . .	222
5.106.2.1 select() . . . . .	222

5.107Translation Class Reference . . . . .	223
5.107.1 Detailed Description . . . . .	224
5.107.2 Member Function Documentation . . . . .	224
5.107.2.1 is_surjective() . . . . .	224
5.108Trap Class Reference . . . . .	224
5.108.1 Detailed Description . . . . .	225
5.108.2 Constructor & Destructor Documentation . . . . .	225
5.108.2.1 Trap() . . . . .	225
5.108.3 Member Function Documentation . . . . .	226
5.108.3.1 get_maximum() . . . . .	226
5.108.3.2 has_known_maximum() . . . . .	226
5.109Umda Class Reference . . . . .	227
5.109.1 Detailed Description . . . . .	228
5.110UniformCrossover Class Reference . . . . .	228
5.110.1 Detailed Description . . . . .	229
5.110.2 Member Function Documentation . . . . .	229
5.110.2.1 breed() . . . . .	229
5.111WalshExpansion Class Reference . . . . .	229
5.111.1 Detailed Description . . . . .	231
5.111.2 Member Function Documentation . . . . .	231
5.111.2.1 random() . . . . .	231
5.112WalshExpansion1 Class Reference . . . . .	231
5.112.1 Detailed Description . . . . .	232
5.112.2 Member Function Documentation . . . . .	233
5.112.2.1 random() . . . . .	233
5.113WalshExpansion2 Class Reference . . . . .	233
5.113.1 Detailed Description . . . . .	234
5.113.2 Member Function Documentation . . . . .	234
5.113.2.1 random() . . . . .	234
5.113.3 Member Data Documentation . . . . .	235
5.113.3.1 _quadratic . . . . .	235
5.114Function::WalshTransformTerm Struct Reference . . . . .	235
5.114.1 Detailed Description . . . . .	236
5.114.2 Member Data Documentation . . . . .	236
5.114.2.1 feature . . . . .	236

# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">hnco</a>	Top-level HNCO namespace . . . . .	13
<a href="#">hnco::algorithm</a>	Algorithms . . . . .	22
<a href="#">hnco::algorithm::bm_pbil</a>	Boltzmann machine PBIL . . . . .	25
<a href="#">hnco::algorithm::eda</a>	Algorithms from the FastEfficientP3 project . . . . .	25
<a href="#">hnco::algorithm::hea</a>	Herding evolutionary algorithm . . . . .	26
<a href="#">hnco::exception</a>	Exceptions . . . . .	26
<a href="#">hnco::function</a>	Functions to be maximized . . . . .	27
<a href="#">hnco::neighborhood</a>	Neighborhoods for local search . . . . .	29
<a href="#">hnco::random</a>	Pseudo random numbers . . . . .	29



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Algorithm . . . . .	38
CompleteSearch . . . . .	57
Hboa . . . . .	90
Ltga . . . . .	119
ParameterLessPopulationPyramid . . . . .	162
IterativeAlgorithm . . . . .	101
BmPbil . . . . .	48
Mimic . . . . .	128
FirstAscentHillClimbing . . . . .	67
GeneticAlgorithm . . . . .	82
Hea< Moment, Herding > . . . . .	91
MuCommaLambdaEa . . . . .	133
MuPlusLambdaEa . . . . .	137
OnePlusLambdaCommaLambdaGa . . . . .	158
PvAlgorithm . . . . .	185
CompactGa . . . . .	55
Mmas . . . . .	130
NpsPbil . . . . .	150
Pbil . . . . .	164
Umda . . . . .	227
RandomLocalSearch . . . . .	190
RandomSearch . . . . .	192
RandomWalk . . . . .	193
Restart . . . . .	195
SimulatedAnnealing . . . . .	198
SteepestAscentHillClimbing . . . . .	210
OnePlusOneEa . . . . .	160
BitHerding . . . . .	45
BitMoment . . . . .	47
Crossover . . . . .	58
BiasedCrossover . . . . .	44
UniformCrossover . . . . .	228
Evaluator	
HncoEvaluator . . . . .	97

ProgressTracker::Event . . . . .	64
Exception . . . . .	64
Error . . . . .	63
LastEvaluation . . . . .	109
PointValueException . . . . .	169
LocalMaximum . . . . .	115
MaximumReached . . . . .	124
TargetReached . . . . .	220
Function . . . . .	71
AbstractLabs . . . . .	31
Labs . . . . .	107
LabsMeritFactor . . . . .	108
AbstractMaxSat . . . . .	32
MaxNae3Sat . . . . .	125
MaxSat . . . . .	126
DeceptiveJump . . . . .	59
EqualProducts . . . . .	61
Factorization . . . . .	65
FourPeaks . . . . .	68
FunctionDecorator . . . . .	76
FunctionController . . . . .	75
Cache . . . . .	52
CallCounter . . . . .	54
OnBudgetFunction . . . . .	152
ProgressTracker . . . . .	179
StopOnMaximum . . . . .	211
StopOnTarget . . . . .	214
FunctionModifier . . . . .	80
AdditiveGaussianNoise . . . . .	34
FunctionMapComposition . . . . .	77
Negation . . . . .	142
PriorNoise . . . . .	177
FunctionPlugin . . . . .	81
Hiff . . . . .	96
Jump . . . . .	105
LeadingOnes . . . . .	109
LinearFunction . . . . .	111
LongPath . . . . .	117
Needle . . . . .	140
NkLandscape . . . . .	148
OneMax . . . . .	155
Plateau . . . . .	167
Qubo . . . . .	187
Ridge . . . . .	196
SixPeaks . . . . .	203
SummationCancellation . . . . .	218
SinusSummationCancellation . . . . .	202
Trap . . . . .	224
WalshExpansion . . . . .	229
WalshExpansion1 . . . . .	231
WalshExpansion2 . . . . .	233
Iterator . . . . .	104
Hypercubeliterator . . . . .	98
NeighborhoodIterator . . . . .	147
HammingSphereIterator . . . . .	88
SingleBitFlipIterator . . . . .	201
LogContext . . . . .	116



ProgressTrackerContext . . . . .	182
Map . . . . .	120
AffineMap . . . . .	36
Injection . . . . .	99
LinearMap . . . . .	113
MapComposition . . . . .	122
Permutation . . . . .	165
Projection . . . . .	183
Translation . . . . .	223
Model . . . . .	131
ModelParameters . . . . .	132
Neighborhood . . . . .	144
MultiBitFlip . . . . .	135
BernoulliProcess . . . . .	41
HammingBall . . . . .	85
HammingSphere . . . . .	87
SingleBitFlip . . . . .	200
Population . . . . .	170
TournamentSelection . . . . .	221
Random . . . . .	190
SpinHerding . . . . .	205
SpinMoment . . . . .	208
StopWatch . . . . .	217
Function::WalshTransformTerm . . . . .	235



## Chapter 3

# Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">AbstractLabs</a>	Abstract class for low autocorrelation binary sequences . . . . .	31
<a href="#">AbstractMaxSat</a>	Abstract class for MaxSat-like functions . . . . .	32
<a href="#">AdditiveGaussianNoise</a>	Additive Gaussian Noise . . . . .	34
<a href="#">AffineMap</a>	Affine map . . . . .	36
<a href="#">Algorithm</a>	Abstract search algorithm . . . . .	38
<a href="#">BernoulliProcess</a>	Bernoulli process . . . . .	41
<a href="#">BiasedCrossover</a>	Biased crossover . . . . .	44
<a href="#">BitHerding</a>	Herding with bit features . . . . .	45
<a href="#">BitMoment</a>	Moment for bit features . . . . .	47
<a href="#">BmPbil</a>	Boltzmann machine PBIL . . . . .	48
<a href="#">Cache</a>	<a href="#">Cache</a> . . . . .	52
<a href="#">CallCounter</a>	Call counter . . . . .	54
<a href="#">CompactGa</a>	Compact genetic algorithm . . . . .	55
<a href="#">CompleteSearch</a>	Complete search . . . . .	57
<a href="#">Crossover</a>	<a href="#">Crossover</a> . . . . .	58
<a href="#">DeceptiveJump</a>	Deceptive jump . . . . .	59
<a href="#">EqualProducts</a>	Equal products . . . . .	61
<a href="#">Error</a>	<a href="#">Error</a> . . . . .	63

<a href="#">ProgressTracker::Event</a>	
<a href="#">Event</a> . . . . .	64
<a href="#">Exception</a>	
Basic exception . . . . .	64
<a href="#">Factorization</a>	
<a href="#">Factorization</a> . . . . .	65
<a href="#">FirstAscentHillClimbing</a>	
First ascent hill climbing . . . . .	67
<a href="#">FourPeaks</a>	
Four Peaks . . . . .	68
<a href="#">Function</a>	
<a href="#">Function</a> . . . . .	71
<a href="#">FunctionController</a>	
Function controller . . . . .	75
<a href="#">FunctionDecorator</a>	
Function decorator . . . . .	76
<a href="#">FunctionMapComposition</a>	
Composition of a function and a map . . . . .	77
<a href="#">FunctionModifier</a>	
Function modifier . . . . .	80
<a href="#">FunctionPlugin</a>	
Function plugin . . . . .	81
<a href="#">GeneticAlgorithm</a>	
Genetic algorithm . . . . .	82
<a href="#">HammingBall</a>	
Hamming ball . . . . .	85
<a href="#">HammingSphere</a>	
Hamming sphere . . . . .	87
<a href="#">HammingSphereIterator</a>	
Hamming sphere neighborhood iterator . . . . .	88
<a href="#">Hboa</a>	
Hierarchical Bayesian Optimization Algorithm . . . . .	90
<a href="#">Hea&lt; Moment, Herding &gt;</a>	
Herding evolutionary algorithm . . . . .	91
<a href="#">Hiff</a>	
Hierarchical if and only if . . . . .	96
<a href="#">HncoEvaluator</a>	
Evaluator for HNCO functions . . . . .	97
<a href="#">HypercubeIterator</a>	
Hypercube iterator . . . . .	98
<a href="#">Injection</a>	
<a href="#">Injection</a> . . . . .	99
<a href="#">IterativeAlgorithm</a>	
Iterative search . . . . .	101
<a href="#">Iterator</a>	
<a href="#">Iterator</a> over bit vectors . . . . .	104
<a href="#">Jump</a>	
<a href="#">Jump</a> . . . . .	105
<a href="#">Labs</a>	
Low autocorrelation binary sequences . . . . .	107
<a href="#">LabsMeritFactor</a>	
Low autocorrelation binary sequences merit factor . . . . .	108
<a href="#">LastEvaluation</a>	
Last evaluation . . . . .	109
<a href="#">LeadingOnes</a>	
Leading ones . . . . .	109
<a href="#">LinearFunction</a>	
Linear function . . . . .	111

<a href="#">LinearMap</a>	
Linear map	113
<a href="#">LocalMaximum</a>	
Local maximum	115
<a href="#">LogContext</a>	
Log context	116
<a href="#">LongPath</a>	
Long path	117
<a href="#">Ltga</a>	
Linkage Tree Genetic Algorithm	119
<a href="#">Map</a>	
Map	120
<a href="#">MapComposition</a>	
Map composition	122
<a href="#">MaximumReached</a>	
Maximum reached	124
<a href="#">MaxNae3Sat</a>	
Max not-all-equal 3SAT	125
<a href="#">MaxSat</a>	
MAX-SAT	126
<a href="#">Mimic</a>	
Mutual information maximizing input clustering	128
<a href="#">Mmas</a>	
Max-min ant system	130
<a href="#">Model</a>	
Model of a Boltzmann machine	131
<a href="#">ModelParameters</a>	
Parameters of a Boltzmann machine	132
<a href="#">MuCommaLambdaEa</a>	
(mu, lambda) EA	133
<a href="#">MultiBitFlip</a>	
Multi bit flip	135
<a href="#">MuPlusLambdaEa</a>	
(mu+lambda) EA	137
<a href="#">Needle</a>	
Needle in a haystack	140
<a href="#">Negation</a>	
Negation	142
<a href="#">Neighborhood</a>	
Neighborhood	144
<a href="#">NeighborhoodIterator</a>	
Neighborhood iterator	147
<a href="#">NkLandscape</a>	
NK landscape	148
<a href="#">NpsPbil</a>	
Population-based incremental learning with negative and positive selection	150
<a href="#">OnBudgetFunction</a>	
CallCounter with a limited number of evaluations	152
<a href="#">OneMax</a>	
OneMax	155
<a href="#">OnePlusLambdaCommaLambdaGa</a>	
(1+(lambda, lambda)) genetic algorithm	158
<a href="#">OnePlusOneEa</a>	
(1+1) EA	160
<a href="#">ParameterLessPopulationPyramid</a>	
Parameter-less Population Pyramid	162
<a href="#">Pbil</a>	
Population-based incremental learning	164

Permutation	
Permutation	165
Plateau	
Plateau	167
PointValueException	
Point-value exception	169
Population	
Population	170
PriorNoise	
Prior noise	177
ProgressTracker	
ProgressTracker	179
ProgressTrackerContext	
Log context for ProgressTracker	182
Projection	
Projection	183
PvAlgorithm	
Probability vector algorithm	185
Qubo	
Quadratic unconstrained binary optimization	187
Random	
Random numbers	190
RandomLocalSearch	
Random local search	190
RandomSearch	
Random search	192
RandomWalk	
Random walk	193
Restart	
Restart	195
Ridge	
Ridge	196
SimulatedAnnealing	
Simulated annealing	198
SingleBitFlip	
One bit neighborhood	200
SingleBitFlipIterator	
Single bit flip neighborhood iterator	201
SinusSummationCancellation	
Summation cancellation with sinus	202
SixPeaks	
Six Peaks	203
SpinHerding	
Herding with spin variables	205
SpinMoment	
Moment for spin variables	208
SteepestAscentHillClimbing	
Steepest ascent hill climbing	210
StopOnMaximum	
Stop on maximum	211
StopOnTarget	
Stop on target	214
StopWatch	
Stop watch	217
SummationCancellation	
Summation cancellation	218
TargetReached	
Target reached	220

TournamentSelection	
Population with tournament selection	221
Translation	
Translation	223
Trap	
Trap	224
Umda	
Univariate marginal distribution algorithm	227
UniformCrossover	
Uniform crossover	228
WalshExpansion	
Walsh expansion	229
WalshExpansion1	
Walsh expansion of degree 1	231
WalshExpansion2	
Walsh expansion of degree 2	233
Function::WalshTransformTerm	
Walsh transform term	235





## Chapter 4

# Namespace Documentation

### 4.1 hnco Namespace Reference

top-level HNCO namespace

#### Namespaces

- [algorithm](#)  
*Algorithms.*
- [exception](#)  
*Exceptions.*
- [function](#)  
*Functions to be maximized.*
- [neighborhood](#)  
*Neighborhoods for local search.*
- [random](#)  
*Pseudo random numbers.*

#### Classes

- class [AffineMap](#)  
*Affine map.*
- class [HypercubeIterator](#)  
*Hypercube iterator.*
- class [Injection](#)  
*Injection.*
- class [Iterator](#)  
*Iterator over bit vectors.*
- class [LinearMap](#)  
*Linear map.*
- class [Map](#)  
*Map.*
- class [MapComposition](#)  
*Map composition.*

- class [Permutation](#)  
*Permutation.*
- class [Projection](#)  
*Projection.*
- class [StopWatch](#)  
*Stop watch.*
- class [Translation](#)  
*Translation.*

## Functions

- template<class T >  
T [square](#) (T x)  
*Generic square function.*
- double [logistic](#) (double x)  
*Logistic function (sigmoid)*

## Types and functions related to bit matrices

- typedef std::vector< [bit\\_vector\\_t](#) > [bit\\_matrix\\_t](#)  
*Bit matrix.*
- void [bm\\_display](#) (const [bit\\_matrix\\_t](#) &M, std::ostream &stream)  
*Display bit matrix.*
- bool [bm\\_is\\_valid](#) (const [bit\\_matrix\\_t](#) &M)  
*Check whether a bit matrix is valid.*
- size\_t [bm\\_num\\_rows](#) (const [bit\\_matrix\\_t](#) &M)  
*Number of rows.*
- size\_t [bm\\_num\\_columns](#) (const [bit\\_matrix\\_t](#) &M)  
*Number of columns.*
- bool [bm\\_is\\_square](#) (const [bit\\_matrix\\_t](#) &M)  
*Check whether the matrix is a square matrix.*
- bool [bm\\_is\\_identity](#) (const [bit\\_matrix\\_t](#) &M)  
*Check whether the matrix is the identity matrix.*
- bool [bm\\_is\\_upper\\_triangular](#) (const [bit\\_matrix\\_t](#) &M)  
*Check whether the matrix is upper triangular.*
- void [bm\\_resize](#) ([bit\\_matrix\\_t](#) &M, std::size\_t num\_rows, std::size\_t num\_columns)  
*Resize a bit matrix.*
- void [bm\\_resize](#) ([bit\\_matrix\\_t](#) &M, std::size\_t num\_rows)  
*Resize a bit matrix and make it a square matrix.*
- void [bm\\_clear](#) ([bit\\_matrix\\_t](#) &M)  
*Clear bit matrix.*
- void [bm\\_identity](#) ([bit\\_matrix\\_t](#) &M)  
*Set the matrix to the identity matrix.*
- void [bm\\_random](#) ([bit\\_matrix\\_t](#) &M)  
*Sample a random bit matrix.*
- void [bm\\_swap\\_rows](#) ([bit\\_matrix\\_t](#) &M, std::size\_t i, std::size\_t j)  
*Swap two rows.*
- void [bm\\_add\\_rows](#) ([bit\\_matrix\\_t](#) &M, std::size\_t i, std::size\_t j)  
*Add two rows.*

- void [bm\\_row\\_echelon\\_form](#) ([bit\\_matrix\\_t](#) &A)  
*Compute a row echelon form of a matrix.*
- [std::size\\_t](#) [bm\\_rank](#) (const [bit\\_matrix\\_t](#) &A)  
*Compute the rank of a matrix.*
- bool [bm\\_solve](#) ([bit\\_matrix\\_t](#) &A, [bit\\_vector\\_t](#) &b)  
*Solve a linear system.*
- bool [bm\\_solve\\_upper\\_triangular](#) ([bit\\_matrix\\_t](#) &A, [bit\\_vector\\_t](#) &b)  
*Solve a linear system in upper triangular form.*
- bool [bm\\_invert](#) ([bit\\_matrix\\_t](#) &M, [bit\\_matrix\\_t](#) &N)  
*Invert a bit matrix.*
- void [bm\\_multiply](#) (const [bit\\_matrix\\_t](#) &M, const [bit\\_vector\\_t](#) &x, [bit\\_vector\\_t](#) &y)  
*Multiply a bit matrix and a bit vector.*
- void [bm\\_transpose](#) (const [bit\\_matrix\\_t](#) &M, [bit\\_matrix\\_t](#) &N)  
*Transpose.*

### Types and functions related to bit

- typedef char [bit\\_t](#)  
*Bit.*
- [bit\\_t](#) [bit\\_flip](#) ([bit\\_t](#) b)  
*Flip bit.*
- [bit\\_t](#) [bit\\_random](#) (double p)  
*Sample a random bit.*

### Types and functions related to bit vectors

- typedef [std::vector](#)< [bit\\_t](#) > [bit\\_vector\\_t](#)  
*Bit vector.*
- typedef [std::pair](#)< [bit\\_vector\\_t](#), double > [point\\_value\\_t](#)  
*Type to represent point value pairs.*
- void [bv\\_display](#) (const [bit\\_vector\\_t](#) &v, [std::ostream](#) &stream)  
*Display bit vector.*
- bool [bv\\_is\\_valid](#) (const [bit\\_vector\\_t](#) &x)  
*Check whether the bit vector is valid.*
- bool [bv\\_is\\_zero](#) (const [bit\\_vector\\_t](#) &x)  
*Check whether the bit vector is zero.*
- int [bv\\_hamming\\_weight](#) (const [bit\\_vector\\_t](#) &x)  
*Hamming weight.*
- int [bv\\_hamming\\_weight](#) (const [std::vector](#)< bool > &x)  
*Hamming weight.*
- int [bv\\_hamming\\_distance](#) (const [bit\\_vector\\_t](#) &x, const [bit\\_vector\\_t](#) &y)  
*Hamming distance between two bit vectors.*
- [bit\\_t](#) [bv\\_dot\\_product](#) (const [bit\\_vector\\_t](#) &x, const [bit\\_vector\\_t](#) &y)  
*Dot product.*
- [bit\\_t](#) [bv\\_dot\\_product](#) (const [bit\\_vector\\_t](#) &x, const [std::vector](#)< bool > &y)  
*Dot product.*
- void [bv\\_clear](#) ([bit\\_vector\\_t](#) &x)  
*Clear bit vector.*
- void [bv\\_flip](#) ([bit\\_vector\\_t](#) &x, [std::size\\_t](#) i)

- *Flip a single bit.*  
void [bv\\_flip](#) ([bit\\_vector\\_t](#) &x, const [bit\\_vector\\_t](#) &mask)
- *Flip many bits.*  
void [bv\\_random](#) ([bit\\_vector\\_t](#) &x)
- *Sample a random bit vector.*  
void [bv\\_random](#) ([bit\\_vector\\_t](#) &x, int k)
- *Sample a random bit vector with given Hamming weight.*  
void [bv\\_add](#) (const [bit\\_vector\\_t](#) &src, [bit\\_vector\\_t](#) &dest)
- *Add two bit vectors.*  
void [bv\\_add](#) (const [bit\\_vector\\_t](#) &x, const [bit\\_vector\\_t](#) &y, [bit\\_vector\\_t](#) &dest)
- *Add two bit vectors.*  
void [bv\\_to\\_vector\\_bool](#) (const [bit\\_vector\\_t](#) &x, std::vector< bool > &y)
- *Convert a bit vector to a bool vector.*  
void [bv\\_from\\_vector\\_bool](#) ([bit\\_vector\\_t](#) &x, const std::vector< bool > &y)
- *Convert a bool vector to a bit vector.*  
std::size\_t [bv\\_to\\_size\\_type](#) (const [bit\\_vector\\_t](#) &x)
- *Convert a bit vector to a size\_t.*  
void [bv\\_from\\_size\\_type](#) ([bit\\_vector\\_t](#) &x, std::size\_t index)
- *Convert a size\_t to a bit vector.*

### Types and functions related to permutations

- typedef std::vector< std::size\_t > [permutation\\_t](#)  
*Permutation type.*
- bool [perm\\_is\\_valid](#) (const [permutation\\_t](#) &permutation)  
*Check that a vector represents a permutation.*
- void [perm\\_identity](#) ([permutation\\_t](#) &s)  
*Identity permutation.*
- void [perm\\_random](#) ([permutation\\_t](#) &s)  
*Sample a random permutation.*

### Types and functions related to sparse bit matrices

- typedef std::vector< [sparse\\_bit\\_vector\\_t](#) > [sparse\\_bit\\_matrix\\_t](#)  
*Sparse bit matrix.*
- void [sbm\\_display](#) (const [sparse\\_bit\\_matrix\\_t](#) &sbm, std::ostream &stream)  
*Display sparse bit matrix.*
- void [bm\\_to\\_sbm](#) (const [bit\\_matrix\\_t](#) &bm, [sparse\\_bit\\_matrix\\_t](#) &sbm)  
*Convert a bit matrix to a sparse bit matrix.*
- void [sbm\\_multiply](#) (const [sparse\\_bit\\_matrix\\_t](#) &M, const [bit\\_vector\\_t](#) &x, [bit\\_vector\\_t](#) &y)  
*Multiply a sparse bit matrix and a bit vector.*

### Types and functions related to sparse bit vectors

- typedef std::vector< std::size\_t > [sparse\\_bit\\_vector\\_t](#)  
*Sparse bit vector.*
- void [bv\\_flip](#) ([bit\\_vector\\_t](#) &x, const [sparse\\_bit\\_vector\\_t](#) &sbv)  
*Flip many bits.*
- void [sbv\\_display](#) (const [sparse\\_bit\\_vector\\_t](#) &v, std::ostream &stream)  
*Display sparse bit vector.*
- void [bv\\_to\\_sbv](#) (const [bit\\_vector\\_t](#) &bv, [sparse\\_bit\\_vector\\_t](#) &sbv)  
*Convert a bit vector to a sparse bit vector.*

### 4.1.1 Detailed Description

top-level HNCV namespace

### 4.1.2 Typedef Documentation

#### 4.1.2.1 bit\_t

```
typedef char bit_t
```

Bit.

A single bit is represented by a char and the values 0 for false and 1 for true.

Definition at line 50 of file bit-vector.hh.

#### 4.1.2.2 sparse\_bit\_matrix\_t

```
typedef std::vector<sparse_bit_vector_t> sparse_bit_matrix_t
```

Sparse bit matrix.

A sparse bit matrix is represented as an array of sparse bit vectors. It knows its number of row, not its number of columns.

Definition at line 45 of file sparse-bit-matrix.hh.

#### 4.1.2.3 sparse\_bit\_vector\_t

```
typedef std::vector<std::size_t> sparse_bit_vector_t
```

Sparse bit vector.

A sparse bit vector is represented as an array containing the indices of its non-zero components. The indices must be sorted in ascending order.

A sparse bit vector does not know the dimension of the space it belongs to.

Definition at line 47 of file sparse-bit-vector.hh.

### 4.1.3 Function Documentation

#### 4.1.3.1 `bm_add_rows()`

```
void bm_add_rows (
    bit_matrix_t & M,
    std::size_t i,
    std::size_t j )
```

Add two rows.

Row  $i$  is added to row  $j$ .

Definition at line 114 of file bit-matrix.cc.

#### 4.1.3.2 `bm_identity()`

```
void bm_identity (
    bit_matrix_t & M )
```

Set the matrix to the identity matrix.

##### Precondition

`bm_is_square(M)`

Definition at line 49 of file bit-matrix.cc.

#### 4.1.3.3 `bm_invert()`

```
bool bm_invert (
    bit_matrix_t & M,
    bit_matrix_t & N )
```

Invert a bit matrix.

##### Parameters

$M$	input matrix
$N$	inverse matrix

##### Precondition

`bm_is_square(M)`  
`bm_is_square(N)`

##### Returns

true if  $M$  is invertible

**Warning**

M is modified by the function. Provided that M is invertible, after returning from the function, M is the identity matrix and N is the computed inverse matrix.

Definition at line 220 of file bit-matrix.cc.

**4.1.3.4 bm\_multiply()**

```
void bm_multiply (
    const bit_matrix_t & M,
    const bit_vector_t & x,
    bit_vector_t & y )
```

Multiply a bit matrix and a bit vector.

The result is  $y = Mx$ .

Definition at line 262 of file bit-matrix.cc.

**4.1.3.5 bm\_rank()**

```
std::size_t bm_rank (
    const bit_matrix_t & A )
```

Compute the rank of a matrix.

**Precondition**

A must be in row echelon form.

Definition at line 153 of file bit-matrix.cc.

**4.1.3.6 bm\_row\_echelon\_form()**

```
void bm_row_echelon_form (
    bit_matrix_t & A )
```

Compute a row echelon form of a matrix.

**Warning**

A is modified by the function.

Definition at line 123 of file bit-matrix.cc.

**4.1.3.7 bm\_solve()**

```
bool bm_solve (
    bit_matrix_t & A,
    bit_vector_t & b )
```

Solve a linear system.

Solve the linear equation  $Ax = b$ .

**Parameters**

$A$	Matrix
$b$	Right hand side

**Precondition**

```
bm_is_square(A)
bm_num_rows(A) == b.size()
```

**Returns**

true if the system has a unique solution

**Warning**

Both  $A$  and  $b$  are modified by the function. Provided that  $A$  is invertible, after returning from the function,  $A$  is the identity matrix and  $b$  is the unique solution to the linear equation.

Definition at line 170 of file bit-matrix.cc.

**4.1.3.8 bm\_solve\_upper\_triangular()**

```
bool bm_solve_upper_triangular (
    bit_matrix_t & A,
    bit_vector_t & b )
```

Solve a linear system in upper triangular form.

Solve the linear equation  $Ax = b$ .

**Parameters**

$A$	Upper triangular matrix
$b$	Right hand side

**Precondition**

```
bm_is_square(A)
bm_num_rows(A) == b.size()
bm_is_upper_triangular(A)
```

**Returns**

true if the system has a unique solution



**Warning**

Both A and b are modified by the function. Provided that A is invertible, after returning from the function, A is the identity matrix and b is the unique solution to the linear equation.

Definition at line 201 of file bit-matrix.cc.

**4.1.3.9 bv\_from\_vector\_bool()**

```
void bv_from_vector_bool (
    bit_vector_t & x,
    const std::vector< bool > & y )
```

Convert a bool vector to a bit vector.

**Warning**

Vectors must be of the same size.

Definition at line 146 of file bit-vector.cc.

**4.1.3.10 bv\_to\_vector\_bool()**

```
void bv_to_vector_bool (
    const bit_vector_t & x,
    std::vector< bool > & y )
```

Convert a bit vector to a bool vector.

**Warning**

Vectors must be of the same size.

Definition at line 133 of file bit-vector.cc.

**4.1.3.11 perm\_identity()**

```
void hnco::perm_identity (
    permutation_t & s ) [inline]
```

Identity permutation.

**Warning**

This function does not set the size of the permutation.

Definition at line 46 of file permutation.hh.

#### 4.1.3.12 perm\_random()

```
void hnco::perm_random (
    permutation_t & s ) [inline]
```

Sample a random permutation.

##### Warning

This function does not set the size of the permutation.

Definition at line 56 of file permutation.hh.

#### 4.1.3.13 sbm\_multiply()

```
void sbm_multiply (
    const sparse_bit_matrix_t & M,
    const bit_vector_t & x,
    bit_vector_t & y )
```

Multiply a sparse bit matrix and a bit vector.

The result is  $y = Mx$ .

Definition at line 47 of file sparse-bit-matrix.cc.

## 4.2 hnco::algorithm Namespace Reference

Algorithms.

### Namespaces

- [bm\\_pbil](#)  
*Boltzmann machine PBIL.*
- [eda](#)  
*Algorithms from the FastEfficientP3 project.*
- [hea](#)  
*Herding evolutionary algorithm.*

## Classes

- class [Algorithm](#)  
*Abstract search algorithm.*
- class [BiasedCrossover](#)  
*Biased crossover.*
- class [CompactGa](#)  
*Compact genetic algorithm.*
- class [CompleteSearch](#)  
*Complete search.*
- class [Crossover](#)  
*Crossover.*
- class [FirstAscentHillClimbing](#)  
*First ascent hill climbing.*
- class [GeneticAlgorithm](#)  
*Genetic algorithm.*
- class [IterativeAlgorithm](#)  
*Iterative search.*
- class [LogContext](#)  
*Log context.*
- class [Mmas](#)  
*Max-min ant system.*
- class [MuCommaLambdaEa](#)  
*(mu, lambda) EA.*
- class [MuPlusLambdaEa](#)  
*(mu+lambda) EA.*
- class [NpsPbil](#)  
*Population-based incremental learning with negative and positive selection.*
- class [OnePlusLambdaCommaLambdaGa](#)  
*(1+(lambda, lambda)) genetic algorithm.*
- class [OnePlusOneEa](#)  
*(1+1) EA.*
- class [Pbil](#)  
*Population-based incremental learning.*
- class [Population](#)  
*Population.*
- class [ProgressTrackerContext](#)  
*Log context for ProgressTracker.*
- class [PvAlgorithm](#)  
*Probability vector algorithm.*
- class [RandomLocalSearch](#)  
*Random local search.*
- class [RandomSearch](#)  
*Random search.*
- class [RandomWalk](#)  
*Random walk.*
- class [Restart](#)  
*Restart.*
- class [SimulatedAnnealing](#)  
*Simulated annealing.*
- class [SteepestAscentHillClimbing](#)

- *Steepest ascent hill climbing.*
- class [TournamentSelection](#)  
*Population with tournament selection.*
- class [Umda](#)  
*Univariate marginal distribution algorithm.*
- class [UniformCrossover](#)  
*Uniform crossover.*

## Functions

- template<class T >  
bool [matrix\\_is\\_symmetric](#) (const std::vector< std::vector< T > > &A)  
*Check for symmetric matrix.*
- template<class T >  
bool [matrix\\_is\\_strictly\\_lower\\_triangular](#) (const std::vector< std::vector< T > > &A)  
*Check for strictly lower triangular matrix.*
- template<class T >  
bool [matrix\\_has\\_diagonal](#) (const std::vector< std::vector< T > > &A, T x)  
*Check for diagonal elements.*
- template<class T >  
bool [matrix\\_has\\_range](#) (const std::vector< std::vector< T > > &A, T inf, T sup)  
*Check for element range.*
- template<class T >  
bool [matrix\\_has\\_dominant\\_diagonal](#) (const std::vector< std::vector< T > > &A)  
*Check for element range.*

## Type and functions related to probability vectors

- typedef std::vector< double > [pv\\_t](#)  
*Probability vector type.*
- double [pv\\_entropy](#) (const [pv\\_t](#) &pv)  
*Entropy of a probability vector.*
- void [pv\\_sample](#) (const [pv\\_t](#) &pv, [bit\\_vector\\_t](#) &x)  
*Sample a bit vector.*
- void [pv\\_uniform](#) ([pv\\_t](#) &pv)  
*Probability vector of the uniform distribution.*
- void [pv\\_init](#) ([pv\\_t](#) &pv)  
*Initialize.*
- void [pv\\_add](#) ([pv\\_t](#) &pv, const [bit\\_vector\\_t](#) &x)  
*Accumulate a bit vector.*
- void [pv\\_add](#) ([pv\\_t](#) &pv, const [bit\\_vector\\_t](#) &x, double weight)  
*Accumulate a bit vector.*
- void [pv\\_average](#) ([pv\\_t](#) &pv, int count)  
*Average.*
- void [pv\\_update](#) ([pv\\_t](#) &pv, double rate, const [bit\\_vector\\_t](#) &x)  
*Update a probability vector toward a bit vector.*
- void [pv\\_update](#) ([pv\\_t](#) &pv, double rate, const std::vector< double > &x)  
*Update a probability vector toward a probability vector.*
- void [pv\\_update](#) ([pv\\_t](#) &pv, double rate, const std::vector< double > &x, const std::vector< double > &y)  
*Update a probability vector toward a probability vector and away from another one.*
- void [pv\\_bound](#) ([pv\\_t](#) &pv, double lower\_bound, double upper\_bound)  
*Bound the components of a probability vector.*

#### 4.2.1 Detailed Description

Algorithms.

### 4.3 `hnco::algorithm::bm_pbil` Namespace Reference

Boltzmann machine PBIL.

#### Classes

- class [BmPbil](#)  
*Boltzmann machine PBIL.*
- class [Model](#)  
*Model of a Boltzmann machine.*
- class [ModelParameters](#)  
*Parameters of a Boltzmann machine.*

#### 4.3.1 Detailed Description

Boltzmann machine PBIL.

### 4.4 `hnco::algorithm::eda` Namespace Reference

Algorithms from the FastEfficientP3 project.

#### Classes

- class [Hboa](#)  
*Hierarchical Bayesian Optimization Algorithm.*
- class [HncoEvaluator](#)  
*Evaluator for HNCO functions.*
- class [Ltga](#)  
*Linkage Tree Genetic Algorithm.*
- class [Mimic](#)  
*Mutual information maximizing input clustering.*
- class [ParameterLessPopulationPyramid](#)  
*Parameter-less Population Pyramid.*

#### 4.4.1 Detailed Description

Algorithms from the FastEfficientP3 project.

## 4.5 hnco::algorithm::hea Namespace Reference

Herding evolutionary algorithm.

### Classes

- class [BitHerding](#)  
*Herding with bit features.*
- struct [BitMoment](#)  
*Moment for bit features.*
- class [Hea](#)  
*Herding evolutionary algorithm.*
- class [SpinHerding](#)  
*Herding with spin variables.*
- struct [SpinMoment](#)  
*Moment for spin variables.*

### 4.5.1 Detailed Description

Herding evolutionary algorithm.

## 4.6 hnco::exception Namespace Reference

Exceptions.

### Classes

- class [Error](#)  
*Error.*
- class [Exception](#)  
*Basic exception.*
- class [LastEvaluation](#)  
*Last evaluation.*
- class [LocalMaximum](#)  
*Local maximum.*
- class [MaximumReached](#)  
*Maximum reached.*
- class [PointValueException](#)  
*Point-value exception.*
- class [TargetReached](#)  
*target reached*

### 4.6.1 Detailed Description

Exceptions.

## 4.7 hnco::function Namespace Reference

Functions to be maximized.

### Classes

- class [AbstractLabs](#)  
*Abstract class for low autocorrelation binary sequences.*
- class [AbstractMaxSat](#)  
*Abstract class for MaxSat-like functions.*
- class [AdditiveGaussianNoise](#)  
*Additive Gaussian Noise.*
- class [Cache](#)  
*Cache.*
- class [CallCounter](#)  
*Call counter.*
- class [DeceptiveJump](#)  
*Deceptive jump.*
- class [EqualProducts](#)  
*Equal products.*
- class [Factorization](#)  
*Factorization.*
- class [FourPeaks](#)  
*Four Peaks.*
- class [Function](#)  
*Function.*
- class [FunctionController](#)  
*Function controller.*
- class [FunctionDecorator](#)  
*Function decorator.*
- class [FunctionMapComposition](#)  
*Composition of a function and a map.*
- class [FunctionModifier](#)  
*Function modifier.*
- class [FunctionPlugin](#)  
*Function plugin.*
- class [Hiff](#)  
*Hierarchical if and only if.*
- class [Jump](#)  
*Jump.*
- class [Labs](#)  
*Low autocorrelation binary sequences.*
- class [LabsMeritFactor](#)  
*Low autocorrelation binary sequences merit factor.*
- class [LeadingOnes](#)  
*Leading ones.*
- class [LinearFunction](#)  
*Linear function.*
- class [LongPath](#)

- Long path.*
- class [MaxNae3Sat](#)
  - Max not-all-equal 3SAT.*
- class [MaxSat](#)
  - MAX-SAT.*
- class [Needle](#)
  - Needle in a haystack.*
- class [Negation](#)
  - Negation.*
- class [NkLandscape](#)
  - NK landscape.*
- class [OnBudgetFunction](#)
  - CallCounter with a limited number of evaluations.*
- class [OneMax](#)
  - OneMax.*
- class [Plateau](#)
  - Plateau.*
- class [PriorNoise](#)
  - Prior noise.*
- class [ProgressTracker](#)
  - ProgressTracker.*
- class [Qubo](#)
  - Quadratic unconstrained binary optimization.*
- class [Ridge](#)
  - Ridge.*
- class [SinusSummationCancellation](#)
  - Summation cancellation with sinus.*
- class [SixPeaks](#)
  - Six Peaks.*
- class [StopOnMaximum](#)
  - Stop on maximum.*
- class [StopOnTarget](#)
  - Stop on target.*
- class [SummationCancellation](#)
  - Summation cancellation.*
- class [Trap](#)
  - Trap.*
- class [WalshExpansion](#)
  - Walsh expansion.*
- class [WalshExpansion1](#)
  - Walsh expansion of degree 1.*
- class [WalshExpansion2](#)
  - Walsh expansion of degree 2.*

## Functions

- `std::ostream & operator<< (std::ostream &stream, const ProgressTracker::Event &event)`
  - Insert formatted output.*
- `bool bv\_is\_locally\_maximal (const bit\_vector\_t &bv, Function &fn, hnco::neighborhood::NeighborhoodIterator &it)`
  - Check whether a bit vector is locally maximal.*
- `bool bv\_is\_globally\_maximal (const bit\_vector\_t &bv, Function &fn)`
  - Check whether a bit vector is globally maximal.*



### 4.7.1 Detailed Description

Functions to be maximized.

## 4.8 `hnco::neighborhood` Namespace Reference

Neighborhoods for local search.

### Classes

- class [BernoulliProcess](#)  
*Bernoulli process.*
- class [HammingBall](#)  
*Hamming ball.*
- class [HammingSphere](#)  
*Hamming sphere.*
- class [HammingSphereIterator](#)  
*Hamming sphere neighborhood iterator.*
- class [MultiBitFlip](#)  
*Multi bit flip.*
- class [Neighborhood](#)  
*Neighborhood.*
- class [NeighborhoodIterator](#)  
*Neighborhood iterator.*
- class [SingleBitFlip](#)  
*One bit neighborhood.*
- class [SingleBitFlipIterator](#)  
*Single bit flip neighborhood iterator.*

### 4.8.1 Detailed Description

Neighborhoods for local search.

There are two unrelated kinds of neighborhoods, those for random local search and those for exhaustive local search.

## 4.9 `hnco::random` Namespace Reference

Pseudo random numbers.

### Classes

- struct [Random](#)  
*Random numbers.*

### 4.9.1 Detailed Description

Pseudo random numbers.



## Chapter 5

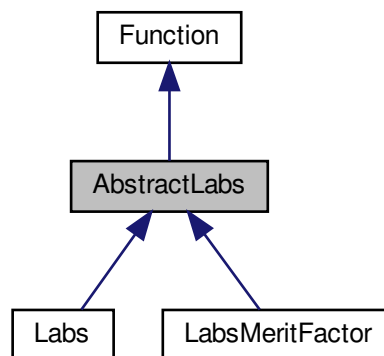
# Class Documentation

### 5.1 AbstractLabs Class Reference

Abstract class for low autocorrelation binary sequences.

```
#include <hnco/functions/labs.hh>
```

Inheritance diagram for AbstractLabs:



#### Public Member Functions

- [AbstractLabs](#) (int n)  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `double` [compute\\_autocorrelation](#) (const `bit_vector_t` &)  
*Compute autocorrelation.*

## Protected Attributes

- `std::vector< int > _sequence`  
*Binary sequence written using 1 and -1.*

### 5.1.1 Detailed Description

Abstract class for low autocorrelation binary sequences.

Definition at line 31 of file labs.hh.

The documentation for this class was generated from the following files:

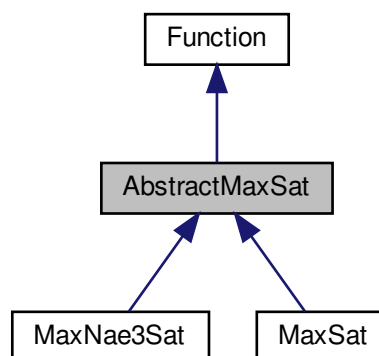
- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

## 5.2 AbstractMaxSat Class Reference

Abstract class for MaxSat-like functions.

```
#include <hnco/functions/max-sat.hh>
```

Inheritance diagram for AbstractMaxSat:



## Public Member Functions

- `AbstractMaxSat ()`  
*Default constructor.*
- `size_t get_bv_size ()`  
*Get bit vector size.*
- `void display (std::ostream &stream)`  
*Display the expression.*
- `virtual void load (std::istream &stream)`  
*Load an instance.*
- `virtual void save (std::ostream &stream)`  
*Save an instance.*

## Protected Attributes

- `std::vector< std::vector< int > > _expression`  
*Expression.*
- `size_t _num_variables`  
*Number of variables.*

### 5.2.1 Detailed Description

Abstract class for MaxSat-like functions.

Definition at line 35 of file max-sat.hh.

### 5.2.2 Member Function Documentation

#### 5.2.2.1 load()

```
void load (
    std::istream & stream ) [virtual]
```

Load an instance.

#### Exceptions

Error	
-------	--

Reimplemented in [MaxNae3Sat](#).

Definition at line 61 of file max-sat.cc.

### 5.2.3 Member Data Documentation

#### 5.2.3.1 \_expression

```
std::vector<std::vector<int> > _expression [protected]
```

Expression.

An expression is represented by a vector of clauses. A clause is represented by a vector of literals. A literal is represented by a non null integer; if the integer is positive then the literal is a variable; if it is negative then it is the logical negation of a variable.

Definition at line 47 of file max-sat.hh.

The documentation for this class was generated from the following files:

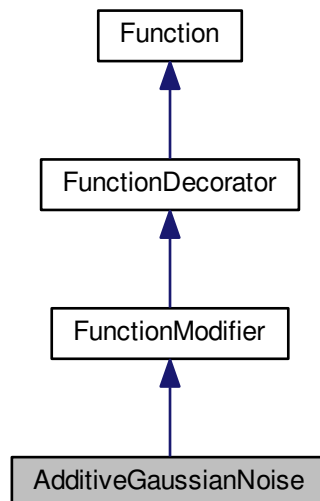
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

## 5.3 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

```
#include <hnco/functions/decorators/function-modifier.hh>
```

Inheritance diagram for AdditiveGaussianNoise:



### Public Member Functions

- [AdditiveGaussianNoise](#) ([Function](#) \*function, double stddev)  
*Constructor.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Information about the function

- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*

### Private Attributes

- std::normal\_distribution< double > [\\_dist](#)  
*Normal distribution.*

## Additional Inherited Members

### 5.3.1 Detailed Description

Additive Gaussian Noise.

Definition at line 176 of file function-modifier.hh.

### 5.3.2 Member Function Documentation

#### 5.3.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Exceptions

<i>Error</i>	
--------------	--

Reimplemented from [Function](#).

Definition at line 198 of file function-modifier.hh.

#### 5.3.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

false

Reimplemented from [Function](#).

Definition at line 202 of file function-modifier.hh.

The documentation for this class was generated from the following files:

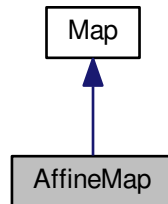
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

## 5.4 AffineMap Class Reference

Affine map.

```
#include <hnco/map.hh>
```

Inheritance diagram for AffineMap:



### Public Member Functions

- void [random](#) (int rows, int cols, bool surjective)  
*Random instance.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- size\_t [get\\_input\\_size](#) ()  
*Get input size.*
- size\_t [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

### Private Member Functions

- template<class Archive >  
void [save](#) (Archive &ar, const unsigned int version) const  
*Save.*
- template<class Archive >  
void [load](#) (Archive &ar, const unsigned int version)  
*Load.*

### Private Attributes

- [bit\\_matrix\\_t \\_bm](#)  
*Bit matrix.*
- [bit\\_vector\\_t \\_bv](#)  
*Translation vector.*



## Friends

- class **boost::serialization::access**

### 5.4.1 Detailed Description

Affine map.

An affine map  $f$  from  $Z_2^m$  to  $Z_2^n$  is defined by  $f(x) = Ax + b$ , where  $A$  is an  $n \times m$  bit matrix and  $b$  is an  $n$ -dimensional bit vector.

Definition at line 257 of file map.hh.

### 5.4.2 Member Function Documentation

#### 5.4.2.1 is\_surjective()

```
bool is_surjective ( ) [virtual]
```

Check for surjective map.

#### Returns

true if  $\text{rank}(\_bm) == \text{bm\_num\_rows}(\_bm)$

Reimplemented from [Map](#).

Definition at line 136 of file map.cc.

#### 5.4.2.2 random()

```
void random (
    int rows,
    int cols,
    bool surjective )
```

Random instance.

#### Parameters

<i>rows</i>	Number of rows
<i>cols</i>	Number of columns
<i>surjective</i>	Flag to ensure a surjective map

## Exceptions

Error
-------

Definition at line 99 of file map.cc.

The documentation for this class was generated from the following files:

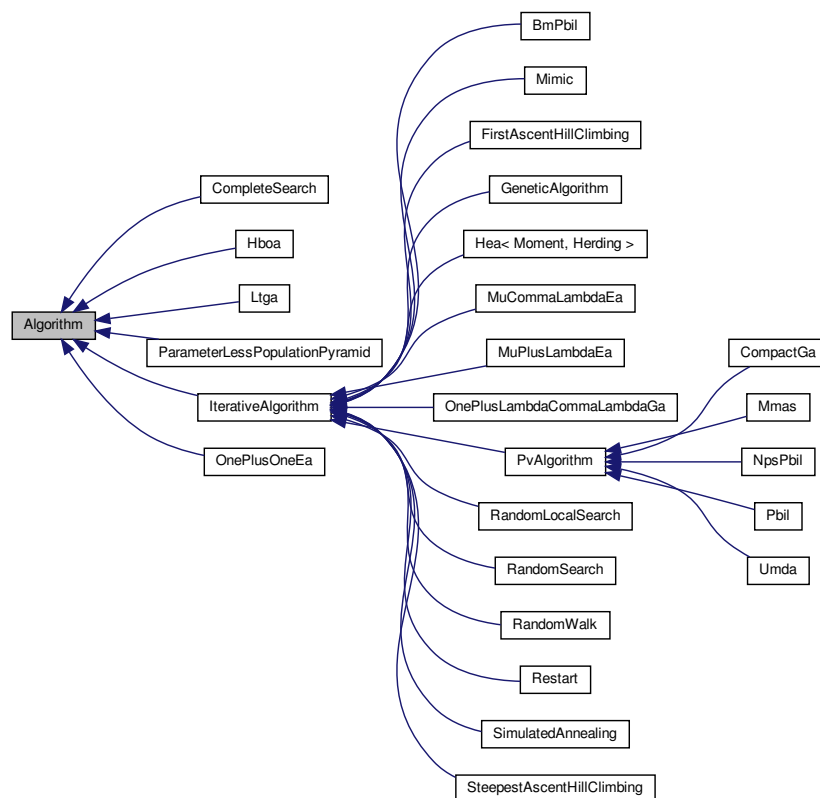
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.5 Algorithm Class Reference

Abstract search algorithm.

```
#include <hnco/algorithms/algorithm.hh>
```

Inheritance diagram for Algorithm:



## Public Member Functions

- [Algorithm](#) (int n)  
*Constructor.*
- virtual [~Algorithm](#) ()  
*Destructor.*

## Optimization

- virtual void [init](#) ()  
*Initialization.*
- virtual void [maximize](#) ()=0  
*Maximize.*

## Getters

- virtual const [point\\_value\\_t](#) & [get\\_solution](#) ()  
*Solution.*
- virtual size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*

## Setters

- virtual void [set\\_function](#) (function::Function \*function)  
*Set function.*
- virtual void [set\\_functions](#) (const std::vector< function::Function \*> functions)  
*Set functions.*
- void [set\\_stream](#) (std::ostream \*x)  
*Output stream.*
- void [set\\_log\\_context](#) (LogContext \*lc)  
*Set log context.*

## Protected Member Functions

### Managing solution

- void [random\\_solution](#) ()  
*Random solution.*
- void [set\\_solution](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Set solution.*
- void [set\\_solution](#) (const [bit\\_vector\\_t](#) &x)  
*Set solution.*
- void [update\\_solution](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Update solution (strict)*
- void [update\\_solution](#) (const [point\\_value\\_t](#) &pv)  
*Update solution (strict)*
- void [update\\_solution](#) (const [bit\\_vector\\_t](#) &x)  
*Update solution (strict).*

## Protected Attributes

- `function::Function * _function`  
*Function.*
- `std::vector< function::Function * > _functions`  
*Functions.*
- `point_value_t _solution`  
*Solution.*
- `LogContext * _log_context = nullptr`  
*Log context.*

## Parameters

- `std::ostream * _stream = &std::cout`  
*Output stream.*

### 5.5.1 Detailed Description

Abstract search algorithm.

All algorithms maximize some given function, sometimes called a fitness function or an objective function.

Definition at line 41 of file algorithm.hh.

### 5.5.2 Member Function Documentation

#### 5.5.2.1 `set_solution()`

```
void set_solution (
    const bit_vector_t & x ) [protected]
```

Set solution.

#### Warning

Evaluates the function once.

Definition at line 47 of file algorithm.cc.

### 5.5.2.2 update\_solution()

```
void update_solution (
    const bit_vector_t & x ) [protected]
```

Update solution (strict).

#### Warning

Evaluates the function once.

Definition at line 70 of file algorithm.cc.

## 5.5.3 Member Data Documentation

### 5.5.3.1 \_functions

```
std::vector<function::Function *> _functions [protected]
```

Functions.

Each thread has its own function.

Definition at line 52 of file algorithm.hh.

The documentation for this class was generated from the following files:

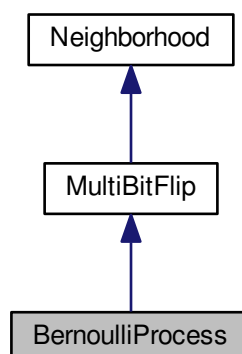
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

## 5.6 BernoulliProcess Class Reference

Bernoulli process.

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for BernoulliProcess:



## Public Member Functions

- [BernoulliProcess](#) (int n)  
*Constructor.*
- [BernoulliProcess](#) (int n, double p)  
*Constructor.*
- void [set\\_probability](#) (double p)  
*Set probability.*

## Private Member Functions

- void [sample\\_bits](#) ()  
*Sample bits.*
- void [bernoulli\\_process](#) ()  
*Bernoulli process.*

## Private Attributes

- std::bernoulli\_distribution [\\_bernoulli\\_dist](#)  
*Bernoulli distribution (biased coin)*
- std::binomial\_distribution< int > [\\_binomial\\_dist](#)  
*Binomial distribution.*
- bool [\\_reservoir\\_sampling](#) = false  
*Reservoir sampling.*

## Parameters

- bool [\\_allow\\_stay](#) = false  
*Allow stay.*
- void [set\\_allow\\_stay](#) (bool x)  
*Set the flag \_allow\_stay.*

## Additional Inherited Members

### 5.6.1 Detailed Description

Bernoulli process.

Each component of the origin bit vector is flipped with some fixed probability. If no component has been flipped at the end, the process is started all over again. Thus the number of flipped bits follows a pseudo binomial law.

Definition at line 220 of file neighborhood.hh.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 BernoulliProcess() [1/2]

```
BernoulliProcess (
    int n ) [inline]
```

Constructor.

## Parameters

$n$	Size of bit vectors
-----	---------------------

The Bernoulli probability is set to  $1 / n$ .

Definition at line 255 of file neighborhood.hh.

## 5.6.2.2 BernoulliProcess() [2/2]

```
BernoulliProcess (
    int n,
    double p ) [inline]
```

Constructor.

## Parameters

$n$	Size of bit vectors
$p$	Bernoulli probability

Definition at line 265 of file neighborhood.hh.

## 5.6.3 Member Function Documentation

## 5.6.3.1 set\_allow\_stay()

```
void set_allow_stay (
    bool x ) [inline]
```

Set the flag `_allow_stay`.

In case no mutation occurs allow the current bit vector to stay unchanged.

Definition at line 292 of file neighborhood.hh.

## 5.6.3.2 set\_probability()

```
void set_probability (
    double p ) [inline]
```

Set probability.

Sets `_reservoir_sampling` to true if  $E(X) < \sqrt{n}$ , where  $X$  is a random variable with a binomial distribution  $B(n, p)$ , that is if  $np < \sqrt{n}$  or  $p < 1 / \sqrt{n}$ .

Definition at line 276 of file neighborhood.hh.

The documentation for this class was generated from the following files:

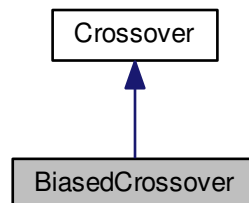
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.7 BiasedCrossover Class Reference

Biased crossover.

```
#include <hnco/algorithms/ea/crossover.hh>
```

Inheritance diagram for BiasedCrossover:



### Public Member Functions

- [BiasedCrossover](#) ()  
*Constructor.*
- void [breed](#) (const [bit\\_vector\\_t](#) &parent1, const [bit\\_vector\\_t](#) &parent2, [bit\\_vector\\_t](#) &offspring)  
*Breed.*
- void [set\\_bias](#) (double b)  
*Set bias.*

### Private Attributes

- `std::bernoulli_distribution` [\\_bernoulli\\_dist](#)  
*Bernoulli distribution.*

#### 5.7.1 Detailed Description

Biased crossover.

Definition at line 75 of file `crossover.hh`.

#### 5.7.2 Member Function Documentation

##### 5.7.2.1 breed()

```
void breed (  
    const bit\_vector\_t & parent1,  
    const bit\_vector\_t & parent2,  
    bit\_vector\_t & offspring ) [virtual]
```

Breed.

Each offspring's bit is copied from second parent with a fixed probability (the crossover bias), from first parent otherwise.



## Parameters

<i>parent1</i>	First parent
<i>parent2</i>	Second parent
<i>offspring</i>	Offspring

Implements [Crossover](#).

Definition at line 45 of file crossover.cc.

The documentation for this class was generated from the following files:

- lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

## 5.8 BitHerding Class Reference

Herding with bit features.

```
#include <hnco/algorithms/hea/bit-herding.hh>
```

### Public Types

- enum { [DYNAMICS\\_MINIMIZE\\_NORM](#), [DYNAMICS\\_MAXIMIZE\\_INNER\\_PRODUCT](#) }

### Public Member Functions

- [BitHerding](#) (int n)  
*Constructor.*
- void [init](#) ()  
*Initialization.*
- void [sample](#) (const [BitMoment](#) &target, [bit\\_vector\\_t](#) &x)  
*Sample a bit vector.*
- double [error](#) (const [BitMoment](#) &target)  
*Compute the error.*

### Getters

- const [BitMoment](#) & [get\\_delta](#) ()  
*Get delta.*

### Setters

- void [set\\_randomize\\_bit\\_order](#) (bool x)  
*Randomize bit order.*
- void [set\\_dynamics](#) (int x)  
*Set the dynamics.*
- void [set\\_weight](#) (double x)  
*Set the weight of second order moments.*

## Protected Member Functions

- void `compute_delta` (const `BitMoment` &target)  
*Compute delta.*
- void `sample_minimize_norm` (const `BitMoment` &target, `bit_vector_t` &x)  
*Sample a bit vector.*
- void `sample_maximize_inner_product` (const `BitMoment` &target, `bit_vector_t` &x)  
*Sample a bit vector.*

## Protected Attributes

- `BitMoment _count`  
*Counter moment.*
- `BitMoment _delta`  
*Delta moment.*
- `permutation_t _permutation`  
*Permutation.*
- `std::uniform_int_distribution< int > _choose_bit`  
*Choose bit.*
- `int _time`  
*Time.*

## Parameters

- `bool _randomize_bit_order` = false  
*Randomize bit order.*
- `int _dynamics` = `DYNAMICS_MINIMIZE_NORM`  
*Dynamics.*
- `double _weight` = 1  
*Weight of second order moments.*

## 5.8.1 Detailed Description

Herding with bit features.

Definition at line 38 of file `bit-herding.hh`.

## 5.8.2 Member Enumeration Documentation

### 5.8.2.1 anonymous enum

anonymous enum

#### Enumerator

<code>DYNAMICS_MINIMIZE_NORM</code>	Dynamics defined as minimization of a norm.
<code>DYNAMICS_MAXIMIZE_INNER_PRODUCT</code>	Dynamics defined as maximization of an inner product.

Definition at line 83 of file bit-herding.hh.

The documentation for this class was generated from the following files:

- lib/hnco/algorithms/hea/bit-herding.hh
- lib/hnco/algorithms/hea/bit-herding.cc

## 5.9 BitMoment Struct Reference

Moment for bit features.

```
#include <hnco/algorithms/hea/bit-moment.hh>
```

### Public Member Functions

- [BitMoment](#) (int n)  
*Constructor.*
- void [uniform](#) ()  
*Set the moment to that of the uniform distribution.*
- void [init](#) ()  
*Initialize.*
- void [add](#) (const [bit\\_vector\\_t](#) &x)  
*Accumulate a bit vector.*
- void [average](#) (int count)  
*Compute average.*
- void [update](#) (const [BitMoment](#) &p, double rate)  
*Update moment.*
- void [bound](#) (double margin)  
*Bound moment.*
- double [distance](#) (const [BitMoment](#) &p) const  
*Distance.*
- double [norm\\_2](#) () const  
*Compute the norm 2.*
- double [diameter](#) () const  
*Compute the diameter.*
- size\_t [size](#) () const  
*Size.*
- void [display](#) (std::ostream &stream)  
*Display.*

### Public Attributes

- std::vector< std::vector< double > > [\\_moment](#)  
*Moment.*
- double [\\_weight](#) = 1  
*Weight of second order moments.*

### 5.9.1 Detailed Description

Moment for bit features.

Definition at line 38 of file bit-moment.hh.

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

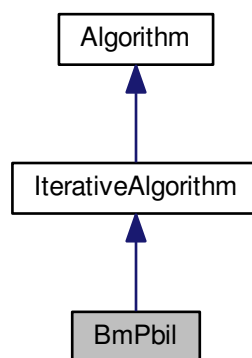
- lib/hnco/algorithms/hea/bit-moment.hh
- lib/hnco/algorithms/hea/bit-moment.cc

## 5.10 BmPbil Class Reference

Boltzmann machine PBIL.

```
#include <hnco/algorithms/bm-pbil/bm-pbil.hh>
```

Inheritance diagram for BmPbil:



### Public Types

- enum { LOG\_NORM\_INFINITE, LOG\_NORM\_L1, **LAST\_LOG** }
- enum { **SAMPLING\_ASYNCHRONOUS**, **SAMPLING\_ASYNCHRONOUS\_FULL\_SCAN**, **SAMPLING\_SYNCHRONOUS** }
- enum { **RESET\_NO\_RESET**, **RESET\_ITERATION**, **RESET\_BIT\_VECTOR** }
- typedef std::bitset< LAST\_LOG > **log\_flags\_t**

### Public Member Functions

- **BmPbil** (int n, int population\_size)  
*Constructor.*
- void **init** ()  
*Initialization.*

## Private Member Functions

- void `iterate ()`  
*Single iteration.*
- void `log ()`  
*Log.*
- void `sample (bit_vector_t &x)`  
*Sample a bit vector.*
- void `sample_asynchronous ()`  
*Asynchronous sampling.*
- void `sample_asynchronous_full_scan ()`  
*Asynchronous sampling with full scan.*
- void `sample_synchronous ()`  
*Synchronous sampling.*

## Private Attributes

- `log_flags_t _log_flags`  
*Log flags.*
- `Population _population`  
*Population.*
- `Model _model`  
*Model.*
- `ModelParameters _parameters_all`  
*Parameters averaged over all individuals.*
- `ModelParameters _parameters_best`  
*Parameters averaged over selected individuals.*
- `ModelParameters _parameters_worst`  
*Parameters averaged over negatively selected individuals.*
- `std::uniform_int_distribution< size_t > _choose_bit`  
*Uniform distribution on bit\_vector\_t components.*
- `permutation_t _permutation`  
*Permutation.*

## Parameters

- `int _selection_size = 1`  
*Selection size (number of selected individuals in the population)*
- `double _learning_rate = 1e-3`  
*Learning rate.*
- `int _num_gs_steps = 100`  
*Number of gibbs sampler steps.*
- `int _num_gs_cycles = 1`  
*Number of gibbs sampler cycles.*
- `bool _negative_positive_selection = false`  
*Negative and positive selection.*
- `int _sampling = SAMPLING_ASYNCHRONOUS`  
*Sampling mode.*
- `int _mc_reset_strategy = RESET_NO_RESET`

- MC reset strategy.*
  - void [set\\_selection\\_size](#) (int x)  
*Set the selection size.*
  - void [set\\_learning\\_rate](#) (double x)  
*Set the learning rate.*
  - void [set\\_num\\_gs\\_steps](#) (int x)  
*Set the number of gibbs sampler steps.*
  - void [set\\_num\\_gs\\_cycles](#) (int x)  
*Set the number of gibbs sampler cycles.*
  - void [set\\_negative\\_positive\\_selection](#) (bool x)  
*Set negative and positive selection.*
  - void [set\\_sampling](#) (int x)  
*Set the sampling mode.*
  - void [set\\_mc\\_reset\\_strategy](#) (int x)  
*Set the MC reset strategy.*
  - void [set\\_log\\_flags](#) (const log\_flags\_t &lf)  
*Set log flags.*

## Additional Inherited Members

### 5.10.1 Detailed Description

Boltzmann machine PBIL.

The BM model is slightly different from the one given in the reference below. More precisely, 0/1 variables are mapped to -1/+1 variables as in Walsh analysis.

Reference:

Arnaud Berny. 2002. Boltzmann machine for population-based incremental learning. In ECAI 2002. IOS Press, Lyon.

Definition at line 51 of file bm-pbil.hh.

### 5.10.2 Member Enumeration Documentation

#### 5.10.2.1 anonymous enum

anonymous enum

#### Enumerator

LOG_NORM_INFINITE	Log infinite norm of the model parameters.
LOG_NORM_L1	Log 1-norm of the model parameters.

Definition at line 56 of file bm-pbil.hh.

### 5.10.2.2 anonymous enum

anonymous enum

#### Enumerator

SAMPLING_ASYNCHRONOUS	Asynchronous sampling. A single component of the internal state is randomly selected then updated by Gibbs sampling. This step is repeated <code>_num_gs_steps</code> times.
SAMPLING_ASYNCHRONOUS_FULL_SCAN	Asynchronous sampling with full scan. To sample a new bit vector, a random permutation is sampled and all components of the internal state are updated by Gibbs sampling in the order defined by the permutation.
SAMPLING_SYNCHRONOUS	Synchronous sampling. The full internal state is updated in one step from the probability vector made of the very marginal probabilities used in Gibbs sampling.

Definition at line 66 of file bm-pbil.hh.

### 5.10.2.3 anonymous enum

anonymous enum

#### Enumerator

RESET_NO_RESET	No reset.
RESET_ITERATION	Reset MC at the beginning of each iteration.
RESET_BIT_VECTOR	Reset MC before sampling each bit vector.

Definition at line 93 of file bm-pbil.hh.

## 5.10.3 Member Function Documentation

### 5.10.3.1 set\_selection\_size()

```
void set_selection_size (
    int x ) [inline]
```

Set the selection size.

The selection size is the number of selected individuals in the population.

Definition at line 210 of file bm-pbil.hh.

The documentation for this class was generated from the following files:

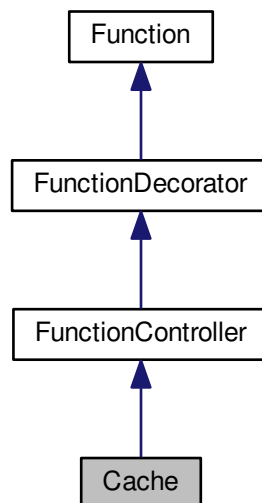
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

## 5.11 Cache Class Reference

[Cache](#).

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for Cache:



### Public Member Functions

- [Cache](#) ([Function](#) \*function)  
*Constructor.*
- bool [provides\\_incremental\\_evaluation](#) ()  
*Check whether the function provides incremental evaluation.*
- double [get\\_lookup\\_ratio](#) ()  
*Get lookup ratio.*

### Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*



## Private Attributes

- `std::unordered_map< std::vector< bool >, double > _cache`  
*Cache.*
- `std::vector< bool > _key`  
*Key.*
- `int _num_evaluations`  
*Evaluation counter.*
- `int _num_lookups`  
*Lookup counter.*

## Additional Inherited Members

### 5.11.1 Detailed Description

`Cache`.

This is a naive approach, in particular with respect to time complexity. Moreover, there is no control on the size of the database.

There is no default hash function for `std::vector<char>` hence the need to first copy a `bit_vector_t` into a `std::vector<bool>`, for which such a function exists, before inserting it or checking its existence in the map.

Definition at line 368 of file `function-controller.hh`.

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 `Cache()`

```
Cache (
    Function * function ) [inline]
```

Constructor.

Parameters

<code>function</code>	Decorated function
-----------------------	--------------------

Definition at line 387 of file `function-controller.hh`.

### 5.11.3 Member Function Documentation

### 5.11.3.1 provides\_incremental\_evaluation()

```
bool provides_incremental_evaluation ( ) [inline], [virtual]
```

Check whether the function provides incremental evaluation.

#### Returns

false

Reimplemented from [FunctionController](#).

Definition at line 396 of file function-controller.hh.

The documentation for this class was generated from the following files:

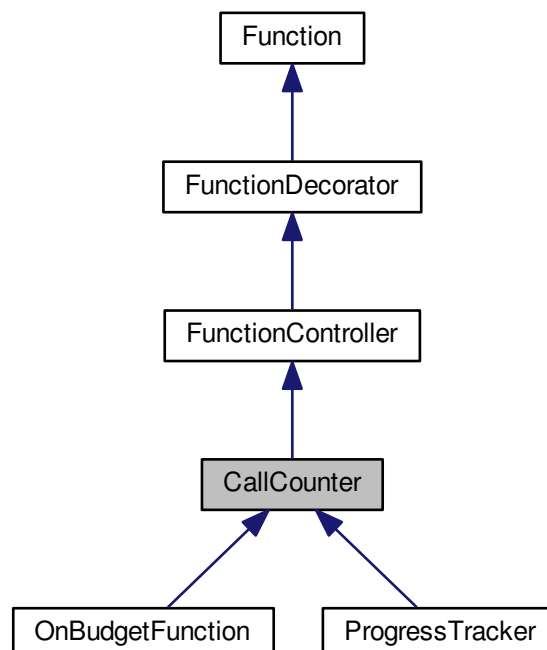
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.12 CallCounter Class Reference

Call counter.

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for CallCounter:



## Public Member Functions

- [CallCounter](#) ([Function](#) \*function)

*Constructor.*

- int [get\\_num\\_calls](#) ()

*Get the number of calls.*

## Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)

*Evaluate a bit vector.*

- double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped←\_bits)

*Incremental evaluation.*

- void [update](#) (const [bit\\_vector\\_t](#) &x, double value)

*Update after a safe evaluation.*

## Protected Attributes

- int [\\_num\\_calls](#)

*Number of calls.*

### 5.12.1 Detailed Description

Call counter.

Definition at line 170 of file function-controller.hh.

The documentation for this class was generated from the following files:

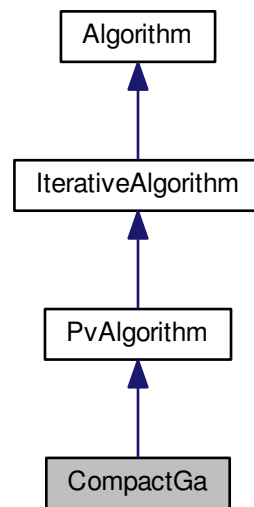
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.13 CompactGa Class Reference

Compact genetic algorithm.

```
#include <hnco/algorithms/pv/compact-ga.hh>
```

Inheritance diagram for CompactGa:



### Public Member Functions

- [CompactGa](#) (int n)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_learning\\_rate](#) (double x)  
*Set the learning rate.*

### Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*

### Protected Attributes

- std::vector< [bit\\_vector\\_t](#) > [\\_candidates](#)  
*Candidates.*

### Parameters

- double [\\_learning\\_rate](#) = 1e-3  
*Learning rate.*

## Additional Inherited Members

### 5.13.1 Detailed Description

Compact genetic algorithm.

Reference:

Georges R. Harik, Fernando G. Lobo, and David E. Goldberg. 1999. The Compact Genetic Algorithm. IEEE Trans. on Evolutionary Computation 3, 4 (November 1999), 287–297.

Definition at line 43 of file compact-ga.hh.

The documentation for this class was generated from the following files:

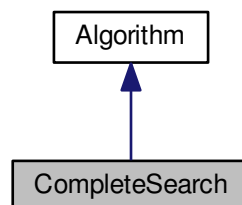
- lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

## 5.14 CompleteSearch Class Reference

Complete search.

```
#include <hnco/algorithms/complete-search.hh>
```

Inheritance diagram for CompleteSearch:



## Public Member Functions

- [CompleteSearch](#) (int n)  
*Constructor.*
- void [maximize](#) ()  
*Maximize.*

## Additional Inherited Members

### 5.14.1 Detailed Description

Complete search.

Definition at line 34 of file complete-search.hh.

The documentation for this class was generated from the following files:

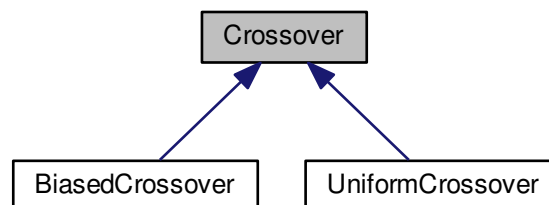
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

## 5.15 Crossover Class Reference

[Crossover](#).

```
#include <hnco/algorithms/ea/crossover.hh>
```

Inheritance diagram for Crossover:



## Public Member Functions

- virtual [~Crossover](#) ()  
*Destructor.*
- virtual void [breed](#) (const [bit\\_vector\\_t](#) &parent1, const [bit\\_vector\\_t](#) &parent2, [bit\\_vector\\_t](#) &offspring)=0  
*Breed.*

### 5.15.1 Detailed Description

[Crossover](#).

Definition at line 35 of file crossover.hh.

## 5.15.2 Member Function Documentation

### 5.15.2.1 breed()

```
virtual void breed (
    const bit\_vector\_t & parent1,
    const bit\_vector\_t & parent2,
    bit\_vector\_t & offspring ) [pure virtual]
```

Breed.

The offspring is the crossover of two parents.

Parameters

<i>parent1</i>	First parent
<i>parent2</i>	Second parent
<i>offspring</i>	Offspring

Implemented in [BiasedCrossover](#), and [UniformCrossover](#).

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

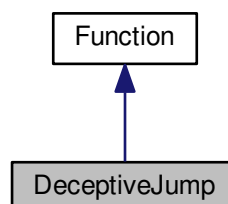
- `lib/hnco/algorithms/ea/crossover.hh`

## 5.16 DeceptiveJump Class Reference

Deceptive jump.

```
#include <hnco/functions/jump.hh>
```

Inheritance diagram for DeceptiveJump:



## Public Member Functions

- [DeceptiveJump](#) (int bv\_size, int gap)  
*Constructor.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- [bool has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [double get\\_maximum](#) ()  
*Get the global maximum.*

## Private Attributes

- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*
- [int \\_gap](#)  
*Gap.*

### 5.16.1 Detailed Description

Deceptive jump.

This is a jump function with a deceptive gap as defined in "Analyzing evolutionary algorithms" by Thomas Jansen, where it is called Jump\_k. Algorithms in the neighborhood of the maximizer (which is the all one bit vector) are taken away from it.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 84 of file jump.hh.

### 5.16.2 Member Function Documentation

#### 5.16.2.1 [get\\_maximum\(\)](#)

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

Returns

[\\_bv\\_size](#) + [\\_gap](#)

Reimplemented from [Function](#).

Definition at line 110 of file jump.hh.



### 5.16.2.2 has\_known\_maximum()

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 106 of file jump.hh.

The documentation for this class was generated from the following files:

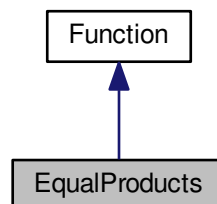
- lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

## 5.17 EqualProducts Class Reference

Equal products.

```
#include <hnco/functions/equal-products.hh>
```

Inheritance diagram for EqualProducts:



### Public Member Functions

- [EqualProducts](#) ()  
*Constructor.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- void [random](#) (int n, double upper\_bound)  
*Random instance.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Private Member Functions

- `template<class Archive >`  
`void serialize (Archive &ar, const unsigned int version)`  
*Serialize.*

## Private Attributes

- `std::vector< double > \_numbers`  
*Numbers.*

## Friends

- class **`boost::serialization::access`**

### 5.17.1 Detailed Description

Equal products.

Partition a finite set of positive numbers into two subsets such that the product of numbers in the first subset is the closest to the product of numbers in the second subset. This is equivalent to the partition problem applied to the logarithms of the given numbers.

The function computes the negation of the distance between the product of numbers corresponding to ones in the bit vector and the product of those corresponding to zeros. The negation is a consequence of the fact that algorithms in HNCO maximize rather than minimize a function.

Reference:

S. Baluja and S. Davies. 1997. Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space. Technical Report CMU- CS-97-107. Carnegie-Mellon University.

Definition at line 61 of file equal-products.hh.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 `random()`

```
void random (
    int n,
    double upper_bound )
```

Random instance.

#### Parameters

<i>n</i>	Size of bit vector
<i>upper_bound</i>	Upper bound of numbers

Definition at line 33 of file equal-products.cc.

The documentation for this class was generated from the following files:

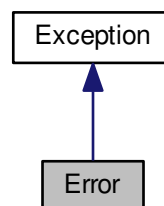
- lib/hnco/functions/equal-products.hh
- lib/hnco/functions/equal-products.cc

## 5.18 Error Class Reference

[Error](#).

```
#include <hnco/exception.hh>
```

Inheritance diagram for Error:



### Public Member Functions

- [Error](#) ()  
*Constructor.*
- [Error](#) (const std::string &s)  
*Constructor.*
- virtual [~Error](#) ()  
*Destructor.*
- virtual const char \* [what](#) () const  
*Get message.*

### Protected Attributes

- std::string [\\_what](#)  
*Message.*

### 5.18.1 Detailed Description

[Error](#).

Definition at line 83 of file exception.hh.

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

- lib/hnco/exception.hh

## 5.19 ProgressTracker::Event Struct Reference

[Event](#).

```
#include <hnco/functions/decorators/function-controller.hh>
```

### Public Attributes

- int [num\\_evaluations](#)  
*Number of evaluations.*
- double [value](#)  
*Value.*

### 5.19.1 Detailed Description

[Event](#).

Definition at line 219 of file function-controller.hh.

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

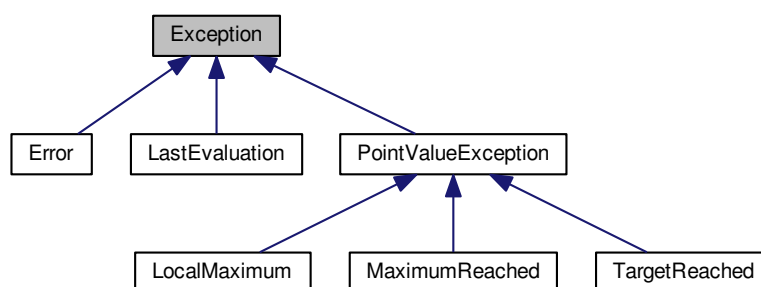
- lib/hnco/functions/decorators/function-controller.hh

## 5.20 Exception Class Reference

Basic exception.

```
#include <hnco/exception.hh>
```

Inheritance diagram for Exception:



### 5.20.1 Detailed Description

Basic exception.

Definition at line 35 of file exception.hh.

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

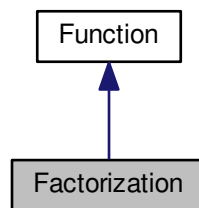
- lib/hnco/exception.hh

## 5.21 Factorization Class Reference

[Factorization](#).

```
#include <hnco/functions/factorization.hh>
```

Inheritance diagram for Factorization:



### Public Member Functions

- [Factorization](#) (std::string path)  
*Constructor.*
- [~Factorization](#) ()  
*Destructor.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- void [display](#) (std::ostream &stream)  
*Display.*
- void [describe](#) (const [bit\\_vector\\_t](#) &x, std::ostream &stream)  
*Describe a bit vector.*

### Private Member Functions

- void [convert](#) (const [bit\\_vector\\_t](#) &x)  
*Convert a bit vector into two numbers.*

## Private Attributes

- [mpz\\_t \\_number](#)  
*Number to factorize.*
- [mpz\\_t \\_first\\_factor](#)  
*First factor.*
- [mpz\\_t \\_second\\_factor](#)  
*Second factor.*
- [mpz\\_t \\_product](#)  
*Product.*
- [std::string \\_first\\_factor\\_string](#)  
*First factor in binary form.*
- [std::string \\_second\\_factor\\_string](#)  
*Secon factor in binary form.*
- [size\\_t \\_number\\_size](#)  
*Number size in bits.*
- [size\\_t \\_first\\_factor\\_size](#)  
*First factor size in bits.*
- [size\\_t \\_second\\_factor\\_size](#)  
*Second factor size in bits.*
- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*

### 5.21.1 Detailed Description

[Factorization.](#)

Reference:

Torbjörn Granlund and the GMP development team. 2012. GNU MP: The GNU Multiple Precision Arithmetic Library (5.0.5 ed.).

<http://gmplib.org/>.

Definition at line 28 of file factorization.hh.

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 Factorization()

```
Factorization (
    std::string path )
```

Constructor.

## Parameters

<i>path</i>	Path to a file containing a number to factorize
-------------	-------------------------------------------------

## Warning

The file is a text file which contains exactly one natural number written in base 10 without any space.

Definition at line 16 of file factorization.cc.

The documentation for this class was generated from the following files:

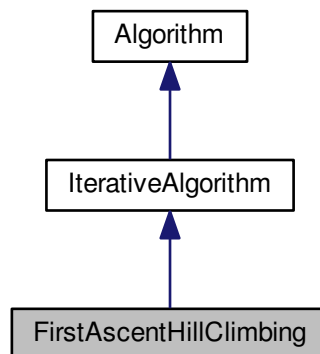
- lib/hnco/functions/factorization.hh
- lib/hnco/functions/factorization.cc

## 5.22 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

```
#include <hnco/algorithms/ls/first-ascent-hill-climbing.hh>
```

Inheritance diagram for FirstAscentHillClimbing:



### Public Member Functions

- [FirstAscentHillClimbing](#) (int n, [neighborhood::NeighborhoodIterator](#) \*neighborhood)  
*Constructor.*
- void [init](#) ()  
*Random initialization.*
- void [init](#) (const [bit\\_vector\\_t](#) &x)  
*Explicit initialization.*
- void [init](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Explicit initialization.*

## Protected Member Functions

- void `iterate` ()  
*Single iteration.*

## Protected Attributes

- `neighborhood::NeighborhoodIterator * _neighborhood`  
*Neighborhood.*

### 5.22.1 Detailed Description

First ascent hill climbing.

Definition at line 35 of file `first-ascent-hill-climbing.hh`.

The documentation for this class was generated from the following files:

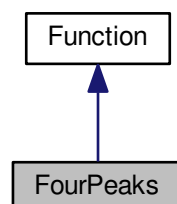
- `lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh`
- `lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc`

## 5.23 FourPeaks Class Reference

Four Peaks.

```
#include <hnco/functions/four-peaks.hh>
```

Inheritance diagram for `FourPeaks`:





## Public Member Functions

- [FourPeaks](#) (int bv\_size, int threshold)  
*Constructor.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- [bool has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [double get\\_maximum](#) ()  
*Get the global maximum.*

## Private Attributes

- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*
- [int \\_threshold](#)  
*Threshold.*
- [int \\_maximum](#)  
*Maximum.*

### 5.23.1 Detailed Description

Four Peaks.

It is defined by

$$f(x) = \max\{\text{head}(x, 1) + \text{tail}(x, 0)\} + R(x)$$

where:

- $\text{head}(x, 1)$  is the length of the longest prefix of  $x$  made of ones;
- $\text{tail}(x, 0)$  is the length of the longest suffix of  $x$  made of zeros;
- $R(x)$  is the reward;
- $R(x) = n$  if  $(\text{head}(x, 1) > t \text{ and } \text{tail}(x, 0) > t)$ ;
- $R(x) = 0$  otherwise;
- the threshold  $t$  is a parameter of the function.

This function has four maxima, of which exactly two are global ones.

For example, if  $n = 6$  and  $t = 1$ :

- $f(111111) = 6$  (local maximum)
- $f(111110) = 5$
- $f(111100) = 10$  (global maximum)

Reference:

S. Baluja and R. Caruana. 1995. Removing the genetics from the standard genetic algorithm. In Proceedings of the 12th Annual Conference on Machine Learning. 38–46.

Definition at line 60 of file four-peaks.hh.

## 5.23.2 Member Function Documentation

### 5.23.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Returns

$2 * \_bv\_size - \_threshold - 1$

Reimplemented from [Function](#).

Definition at line 91 of file four-peaks.hh.

### 5.23.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 87 of file four-peaks.hh.

The documentation for this class was generated from the following files:

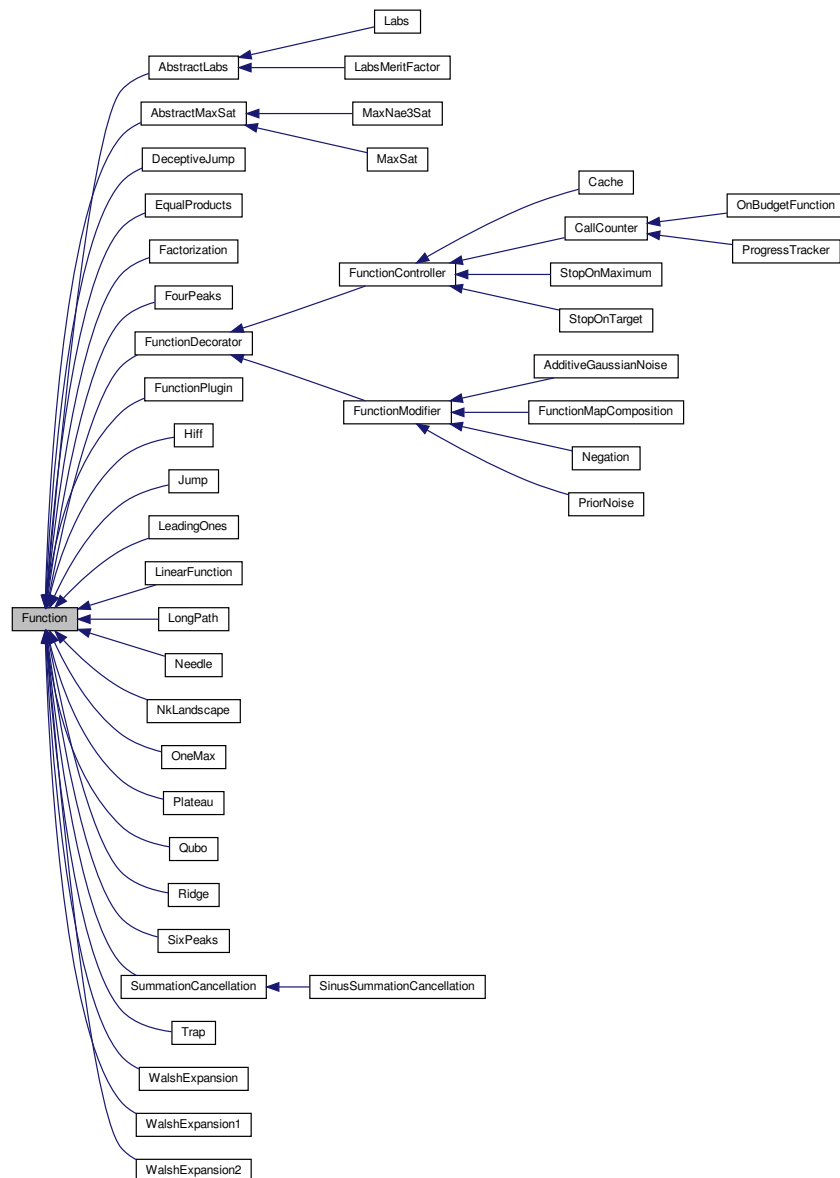
- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

## 5.24 Function Class Reference

[Function.](#)

```
#include <hnco/functions/function.hh>
```

Inheritance diagram for Function:



### Classes

- struct [WalshTransformTerm](#)  
*Walsh transform term.*

## Public Member Functions

- virtual [~Function](#) ()

*Destructor.*

### Information about the function

- virtual size\_t [get\\_bv\\_size](#) ()=0  
*Get bit vector size.*
- virtual double [get\\_maximum](#) ()  
*Get the global maximum.*
- virtual bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- virtual bool [provides\\_incremental\\_evaluation](#) ()  
*Check whether the function provides incremental evaluation.*
- virtual void [compute\\_walsh\\_transform](#) (std::vector< [Function::WalshTransformTerm](#) > &terms)  
*Compute the Walsh transform of the function.*

### Evaluation

- virtual double [eval](#) (const [bit\\_vector\\_t](#) &)=0  
*Evaluate a bit vector.*
- virtual double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped\_bits)  
*Incremental evaluation.*
- virtual double [safe\\_eval](#) (const [bit\\_vector\\_t](#) &x)  
*Safely evaluate a bit vector.*
- virtual void [update](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Update after a safe evaluation.*

### Display

- virtual void [display](#) (std::ostream &stream)  
*Display.*
- virtual void [describe](#) (const [bit\\_vector\\_t](#) &x, std::ostream &stream)  
*Describe a bit vector.*

## 5.24.1 Detailed Description

[Function](#).

Definition at line 40 of file `function.hh`.

## 5.24.2 Member Function Documentation

### 5.24.2.1 `compute_walsh_transform()`

```
void compute_walsh_transform (
    std::vector< Function::WalshTransformTerm > & terms ) [virtual]
```

Compute the Walsh transform of the function.

Let  $f$  be a fitness function defined on the hypercube  $\{0, 1\}^n$ . Then it can be expressed as  $\sum_u c_u \chi_u$  where  $c_u = \langle f, \chi_u \rangle$ ,  $\langle f, g \rangle = \frac{1}{2^n} \sum_x f(x)g(x)$ ,  $\chi_u(x) = (-1)^{x \cdot u}$ , and  $x \cdot u = \sum_i x_i u_i \pmod{2}$ . In the respective sums, we have  $x$  and  $u$  in the hypercube and  $i$  in  $\{1, \dots, n\}$ .

We have dropped the normalizing constant  $2^n$  since we are mostly interested in ratios  $|c_u/c_{\max}|$ , where  $c_{\max}$  is the coefficient with the largest amplitude.

## Parameters

<i>terms</i>	Vector of non zero terms of the Walsh transform
--------------	-------------------------------------------------

## Warning

The time complexity is exponential in the dimension  $n$ . The computation is done with two nested loops over the hypercube. It requires  $2^n$  function evaluations and  $2^{2n}$  dot products and additions.

The size of the Walsh transform is potentially exponential in the dimension  $n$ . For example, if  $n = 10$  then the number of terms is at most 1024.

Definition at line 31 of file function.cc.

5.24.2.2 `get_maximum()`

```
virtual double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

## Exceptions

<i>Error</i>	
--------------	--

Reimplemented in [Plateau](#), [Ridge](#), [AdditiveGaussianNoise](#), [Hiff](#), [SixPeaks](#), [Needle](#), [FunctionMapComposition](#), [LeadingOnes](#), [DeceptiveJump](#), [LongPath](#), [FourPeaks](#), [SummationCancellation](#), [Trap](#), [LinearFunction](#), [Negation](#), [PriorNoise](#), [Jump](#), [OneMax](#), and [FunctionController](#).

Definition at line 79 of file function.hh.

5.24.2.3 `incremental_eval()`

```
virtual double incremental_eval (
    const bit\_vector\_t & x,
    double value,
    const hnco::sparse\_bit\_vector\_t & flipped_bits ) [inline], [virtual]
```

Incremental evaluation.

## Exceptions

<i>Error</i>	
--------------	--

Reimplemented in [OnBudgetFunction](#), [ProgressTracker](#), [CallCounter](#), [StopOnTarget](#), [StopOnMaximum](#), [Negation](#), and [OneMax](#).

Definition at line 132 of file function.hh.

#### 5.24.2.4 provides\_incremental\_evaluation()

```
virtual bool provides_incremental_evaluation ( ) [inline], [virtual]
```

Check whether the function provides incremental evaluation.

##### Returns

false

Reimplemented in [Cache](#), [Negation](#), [PriorNoise](#), [OneMax](#), and [FunctionController](#).

Definition at line 87 of file function.hh.

#### 5.24.2.5 safe\_eval()

```
virtual double safe_eval (
    const bit\_vector\_t & x ) [inline], [virtual]
```

Safely evaluate a bit vector.

Must be thread-safe, that is must avoid throwing exceptions and updating global states (e.g. maximum) in function decorators.

Reimplemented in [FunctionController](#).

Definition at line 142 of file function.hh.

The documentation for this class was generated from the following files:

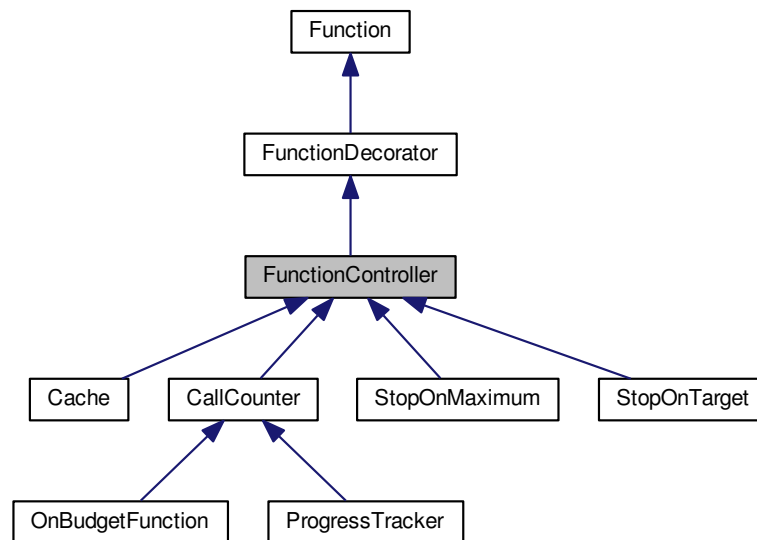
- [lib/hnco/functions/function.hh](#)
- [lib/hnco/functions/function.cc](#)

## 5.25 FunctionController Class Reference

Function controller.

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for FunctionController:



### Public Member Functions

- [FunctionController](#) ([Function](#) \*function)  
*Constructor.*

### Information about the function

- [size\\_t](#) [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double](#) [get\\_maximum](#) ()  
*Get the global maximum.*
- [bool](#) [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [bool](#) [provides\\_incremental\\_evaluation](#) ()  
*Check whether the function provides incremental evaluation.*

### Evaluation

- [double](#) [safe\\_eval](#) (const [bit\\_vector\\_t](#) &x)  
*Safely evaluate a bit vector.*

## Additional Inherited Members

### 5.25.1 Detailed Description

[Function](#) controller.

Definition at line 39 of file function-controller.hh.

### 5.25.2 Member Function Documentation

#### 5.25.2.1 provides\_incremental\_evaluation()

```
bool provides_incremental_evaluation ( ) [inline], [virtual]
```

Check whether the function provides incremental evaluation.

#### Returns

true if the decorated function does

Reimplemented from [Function](#).

Reimplemented in [Cache](#).

Definition at line 64 of file function-controller.hh.

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

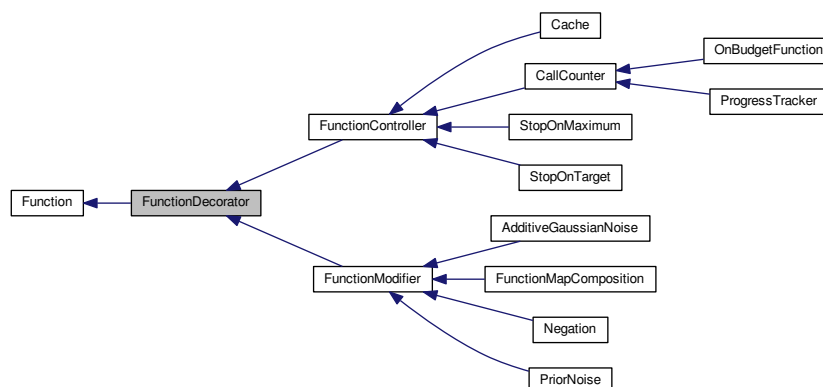
- lib/hnco/functions/decorators/function-controller.hh

## 5.26 FunctionDecorator Class Reference

[Function](#) decorator.

```
#include <hnco/functions/decorators/function-decorator.hh>
```

Inheritance diagram for FunctionDecorator:





### Public Member Functions

- [FunctionDecorator](#) ([Function](#) \*function)  
*Constructor.*

### Display

- void [display](#) (std::ostream &stream)  
*Display.*
- void [describe](#) (const [bit\\_vector\\_t](#) &x, std::ostream &stream)  
*Describe a bit vector.*

### Protected Attributes

- [Function](#) \* [\\_function](#)  
*Decorated function.*

#### 5.26.1 Detailed Description

[Function](#) decorator.

Definition at line 37 of file `function-decorator.hh`.

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

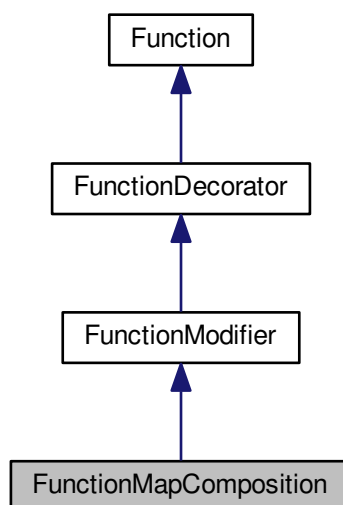
- `lib/hnco/functions/decorators/function-decorator.hh`

## 5.27 FunctionMapComposition Class Reference

Composition of a function and a map.

```
#include <hnco/functions/decorators/function-modifier.hh>
```

Inheritance diagram for `FunctionMapComposition`:



## Public Member Functions

- [FunctionMapComposition](#) ([Function](#) \*function, [Map](#) \*map)  
*Constructor.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Information about the function

- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*

## Display

- void [describe](#) (const [bit\\_vector\\_t](#) &x, std::ostream &stream)  
*Describe a bit vector.*

## Private Attributes

- [Map](#) \* [\\_map](#)  
*Map.*
- [bit\\_vector\\_t](#) [\\_bv](#)  
*Image of bit vectors under the map.*

## Additional Inherited Members

### 5.27.1 Detailed Description

Composition of a function and a map.

Definition at line 106 of file function-modifier.hh.

### 5.27.2 Constructor & Destructor Documentation

#### 5.27.2.1 FunctionMapComposition()

```
FunctionMapComposition (
    Function * function,
    Map * map ) [inline]
```

Constructor.

#### Precondition

```
map->get_output_size() == function->get_bv_size()
```

### Exceptions

Error
-------

Definition at line 121 of file function-modifier.hh.

## 5.27.3 Member Function Documentation

### 5.27.3.1 get\_maximum()

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

### Exceptions

Error
-------

Reimplemented from [Function](#).

Definition at line 141 of file function-modifier.hh.

### 5.27.3.2 has\_known\_maximum()

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

### Returns

true if the function has a known maximum and the map is bijective.

Reimplemented from [Function](#).

Definition at line 151 of file function-modifier.hh.

The documentation for this class was generated from the following files:

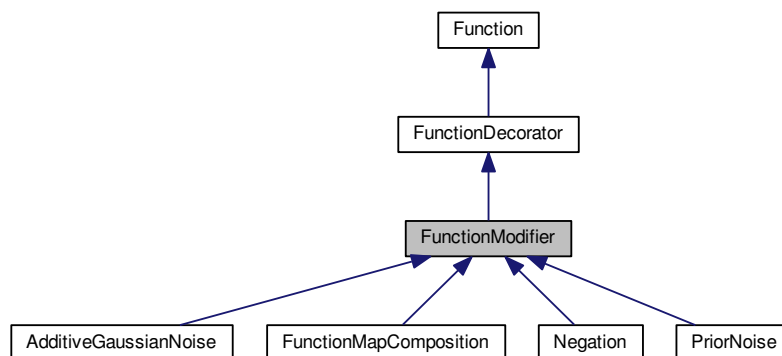
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

## 5.28 FunctionModifier Class Reference

Function modifier.

```
#include <hnco/functions/decorators/function-modifier.hh>
```

Inheritance diagram for FunctionModifier:



### Public Member Functions

- `FunctionModifier (Function *function)`  
*Constructor.*

### Additional Inherited Members

#### 5.28.1 Detailed Description

Function modifier.

Definition at line 37 of file `function-modifier.hh`.

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

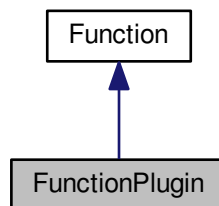
- `lib/hnco/functions/decorators/function-modifier.hh`

## 5.29 FunctionPlugin Class Reference

Function plugin.

```
#include <hnco/functions/plugin.hh>
```

Inheritance diagram for FunctionPlugin:



### Public Member Functions

- [FunctionPlugin](#) (int bv\_size, std::string path, std::string name)  
*Constructor.*
- [~FunctionPlugin](#) ()  
*Destructor.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Private Types

- typedef double(\* [extern\\_function\\_t](#)) (const char[], size\_t)  
*Type of an extern function.*

### Private Attributes

- size\_t [\\_bv\\_size](#)  
*Bit vector size.*
- void \* [\\_handle](#)  
*Handle returned by dlopen.*
- [extern\\_function\\_t](#) [\\_extern\\_function](#)  
*Extern function.*

### 5.29.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

### 5.29.2 Constructor & Destructor Documentation

#### 5.29.2.1 FunctionPlugin()

```
FunctionPlugin (
    int bv_size,
    std::string path,
    std::string name )
```

Constructor.

#### Parameters

<i>bv_size</i>	Size of bit vectors
<i>path</i>	Path to a shared library
<i>name</i>	Name of a function of the shared library

Definition at line 35 of file plugin.cc.

The documentation for this class was generated from the following files:

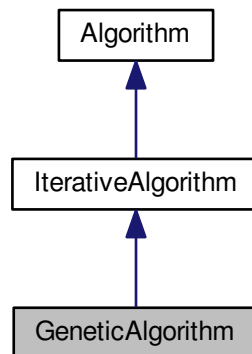
- lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

## 5.30 GeneticAlgorithm Class Reference

Genetic algorithm.

```
#include <hnco/algorithms/ea/genetic-algorithm.hh>
```

Inheritance diagram for GeneticAlgorithm:



### Public Member Functions

- [GeneticAlgorithm](#) (int n, int mu)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_mutation\\_probability](#) (double x)  
*Set the mutation probability.*
- void [set\\_crossover\\_probability](#) (double x)  
*Set the crossover probability.*
- void [set\\_tournament\\_size](#) (int x)  
*Set the tournament size.*
- void [set\\_allow\\_stay](#) (bool x)  
*Set the flag \_allow\_stay.*

### Private Member Functions

- void [iterate](#) ()  
*Single iteration.*

### Private Attributes

- [TournamentSelection \\_parents](#)  
*Parents.*
- [TournamentSelection \\_offsprings](#)  
*Offsprings.*
- [neighborhood::BernoulliProcess \\_mutation](#)

*Mutation operator.*

- `std::bernoulli_distribution _do_crossover`

*Do crossover.*

- `UniformCrossover _crossover`

*Uniform crossover.*

## Parameters

- `double _mutation_probability`  
*Mutation probability.*
- `double _crossover_probability = 0.5`  
*Crossover probability.*
- `int _tournament_size = 10`  
*Tournament size.*
- `bool _allow_stay = false`  
*Allow stay.*

## Additional Inherited Members

### 5.30.1 Detailed Description

Genetic algorithm.

- Tournament selection for reproduction
- Uniform crossover
- Mutation
- (mu, mu) selection (offspring population replaces parent population)

Reference:

J. H. Holland. 1975. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

Definition at line 51 of file genetic-algorithm.hh.

### 5.30.2 Constructor & Destructor Documentation

#### 5.30.2.1 GeneticAlgorithm()

```
GeneticAlgorithm (
    int n,
    int mu ) [inline]
```

Constructor.



## Parameters

<i>n</i>	Size of bit vectors
<i>mu</i>	Population size

Definition at line 97 of file genetic-algorithm.hh.

### 5.30.3 Member Function Documentation

#### 5.30.3.1 set\_allow\_stay()

```
void set_allow_stay (
    bool x ) [inline]
```

Set the flag `_allow_stay`.

In case no mutation occurs allow the current bit vector to stay unchanged.

Definition at line 125 of file genetic-algorithm.hh.

The documentation for this class was generated from the following files:

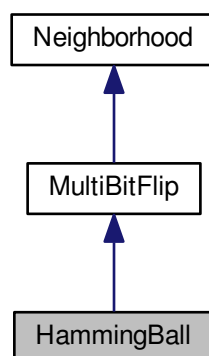
- lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

## 5.31 HammingBall Class Reference

Hamming ball.

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for HammingBall:



## Public Member Functions

- [HammingBall](#) (int n, int r)  
*Constructor.*

## Private Member Functions

- void [sample\\_bits](#) ()  
*Sample bits.*

## Private Attributes

- int [\\_radius](#)  
*Radius of the ball.*
- `std::uniform_int_distribution< int > \_choose\_k`  
*Choose the distance to the center.*

## Additional Inherited Members

### 5.31.1 Detailed Description

Hamming ball.

Choose k uniformly on [1..r], where r is the radius of the ball, choose k bits uniformly among n and flip them.

Definition at line 304 of file neighborhood.hh.

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 HammingBall()

```
HammingBall (
    int n,
    int r ) [inline]
```

Constructor.

#### Parameters

<i>n</i>	Size of bit vectors
<i>r</i>	Radius of the ball

Definition at line 323 of file neighborhood.hh.

The documentation for this class was generated from the following files:

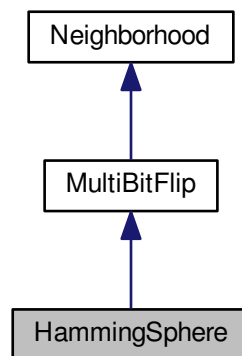
- `lib/hnco/neighborhoods/neighborhood.hh`
- `lib/hnco/neighborhoods/neighborhood.cc`

## 5.32 HammingSphere Class Reference

Hamming sphere.

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for HammingSphere:



### Public Member Functions

- `HammingSphere` (int n, int r)  
*Constructor.*
- void `set_radius` (int r)  
*Set radius.*

### Private Member Functions

- void `sample_bits` ()  
*Sample bits.*

### Private Attributes

- int `_radius`  
*Radius of the sphere.*

## Additional Inherited Members

### 5.32.1 Detailed Description

Hamming sphere.

Uniformly choose  $r$  bits among  $n$  and flip them, where  $r$  is the radius of the sphere.

Definition at line 341 of file neighborhood.hh.

### 5.32.2 Constructor & Destructor Documentation

#### 5.32.2.1 HammingSphere()

```
HammingSphere (
    int n,
    int r ) [inline]
```

Constructor.

#### Parameters

$n$	Size of bit vectors
$r$	Radius of the sphere

Definition at line 357 of file neighborhood.hh.

The documentation for this class was generated from the following files:

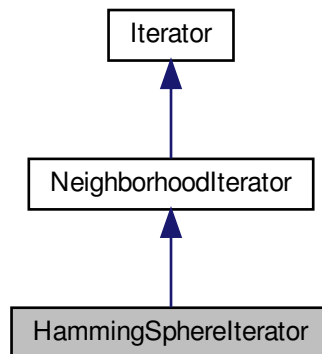
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.33 HammingSphereIterator Class Reference

Hamming sphere neighborhood iterator.

```
#include <hnco/neighborhoods/neighborhood-iterator.hh>
```

Inheritance diagram for HammingSphereIterator:



### Public Member Functions

- [HammingSphereIterator](#) (int n, int r)  
*Constructor.*
- bool [has\\_next](#) ()  
*Has next bit vector.*
- const [bit\\_vector\\_t](#) & [next](#) ()  
*Next bit vector.*

### Private Attributes

- [bit\\_vector\\_t](#) [\\_mask](#)  
*Mutation mask.*
- int [\\_radius](#)  
*Radius of the ball.*
- int [\\_index](#)  
*Index of the next bit to shift to the right.*
- int [\\_weight](#)  
*Partial Hamming weight.*

### Additional Inherited Members

#### 5.33.1 Detailed Description

Hamming sphere neighborhood iterator.

This iterator enumerates mutation masks with hamming weight equal to the given radius. Suppose that `_mask` has a first (from left to right) sequence of ones of length `_weight` and ending at `_index`:

0 ... 0 1 ... 1 0 ...

Then the next mask is obtained by moving to the left the first `_weight - 1` ones and moving to the right the last one.

1 ... 1 0 ... 0 1 ...

Definition at line 91 of file neighborhood-iterator.hh.

### 5.33.2 Constructor & Destructor Documentation

#### 5.33.2.1 HammingSphereIterator()

```
HammingSphereIterator (
    int  $n$ ,
    int  $r$  ) [inline]
```

Constructor.

#### Parameters

$n$	Size of bit vectors
$r$	Radius of Hamming Ball

Definition at line 113 of file neighborhood-iterator.hh.

The documentation for this class was generated from the following files:

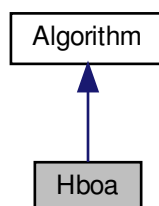
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

## 5.34 Hboa Class Reference

Hierarchical Bayesian Optimization Algorithm.

```
#include <hnco/algorithms/eda/hboa.hh>
```

Inheritance diagram for Hboa:



## Public Member Functions

- [Hboa](#) (int n)  
*Constructor.*
- void [maximize](#) ()  
*Maximize.*
- void [set\\_population\\_size](#) (int n)  
*Set population size.*

## Private Attributes

- int [\\_population\\_size](#) = 10  
*Population size.*

## Additional Inherited Members

### 5.34.1 Detailed Description

Hierarchical Bayesian Optimization Algorithm.

Implementation of the Hierarchical Bayesian Optimization Algorithm and helper classes based on the publication: Pelikan, M. and Goldberg, D. (2006). Hierarchical bayesian optimization algorithm. In Scalable Optimization via Probabilistic Modeling, volume 33 of Studies in Computational Intelligence, pages 63–90. Springer Berlin Heidelberg.

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 42 of file hboa.hh.

The documentation for this class was generated from the following files:

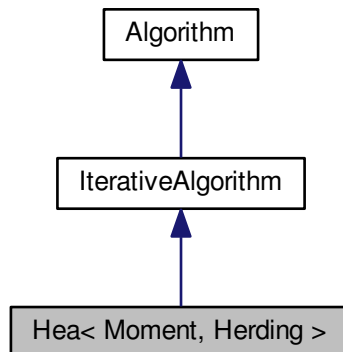
- lib/hnco/algorithms/eda/hboa.hh
- lib/hnco/algorithms/eda/hboa.cc

## 5.35 Hea< Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

```
#include <hnco/algorithms/hea/hea.hh>
```

Inheritance diagram for Hea< Moment, Herding >:



## Public Types

- enum {  
[LOG\\_ERROR](#), [LOG\\_DTU](#), [LOG\\_DELTA](#), [LOG\\_SELECTION](#),  
[LOG\\_MOMENT\\_MATRIX](#), [LAST\\_LOG](#) }
- typedef std::bitset< [LAST\\_LOG](#) > [log\\_flags\\_t](#)  
*Type for log flags.*

## Public Member Functions

- [Hea](#) (int n, int population\_size)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

## Setters

- void [set\\_herding](#) (Herding \*x)  
*Set the herding algorithm.*
- void [set\\_margin](#) (double x)  
*Set the moment margin.*
- void [set\\_selection\\_size](#) (int x)  
*Set the selection size.*
- void [set\\_reset\\_period](#) (int x)  
*Set the reset period.*
- void [set\\_learning\\_rate](#) (double x)  
*Set the learning rate.*
- void [set\\_bound\\_moment](#) (bool x)  
*Set the bound moment after update.*
- void [set\\_weight](#) (double weight)  
*Set weight.*
- void [set\\_log\\_flags](#) (const [log\\_flags\\_t](#) &lf)  
*Set log flags.*



## Private Member Functions

- void [iterate](#) ()  
*Single iteration.*
- void [log](#) ()  
*Log.*

## Private Attributes

- Moment [\\_target](#)  
*Moment.*
- Moment [\\_selection](#)  
*Moment of selected individuals.*
- Moment [\\_uniform](#)  
*Uniform moment.*
- [algorithm::Population](#) [\\_population](#)  
*Population.*
- Herding \* [\\_herding](#)  
*Herding.*

## Logging

- double [\\_error\\_cache](#)  
*Error cache.*
- double [\\_dtu\\_cache](#)  
*Distance to uniform cache.*
- double [\\_delta\\_cache](#)  
*Delta cache.*
- double [\\_selection\\_cache](#)  
*Selection distance cache.*
- [log\\_flags\\_t](#) [\\_log\\_flags](#)  
*Log flags.*

## Parameters

- double [\\_margin](#)  
*Moment margin.*
- int [\\_selection\\_size](#) = 1  
*Selection size.*
- int [\\_reset\\_period](#) = 0  
*Reset period.*
- double [\\_learning\\_rate](#) = 1e-4  
*Learning rate.*
- bool [\\_bound\\_moment](#) = false  
*Bound moment after update.*

## Additional Inherited Members

### 5.35.1 Detailed Description

```
template<class Moment, class Herding>
class hnco::algorithm::hea::Hea< Moment, Herding >
```

Herding evolutionary algorithm.

Reference:

Arnaud Berny. 2015. Herding Evolutionary Algorithm. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15). ACM, New York, NY, USA, 1355–1356.

Definition at line 48 of file hea.hh.

### 5.35.2 Member Enumeration Documentation

#### 5.35.2.1 anonymous enum

```
anonymous enum
```

Enumerator

LOG_ERROR	Log error.
LOG_DTU	Log distance to uniform.
LOG_DELTA	Log delta (moment increment)
LOG_SELECTION	Log the distance between the target and the selection moment.
LOG_MOMENT_MATRIX	Log the moment matrix.

Definition at line 53 of file hea.hh.

### 5.35.3 Constructor & Destructor Documentation

#### 5.35.3.1 Hea()

```
Hea (
    int n,
    int population_size ) [inline]
```

Constructor.

**Parameters**

$n$	Size of bit vectors
-----	---------------------

`_margin` is initialized to  $1 / n$ .

Definition at line 211 of file hea.hh.

## 5.35.4 Member Function Documentation

### 5.35.4.1 `set_reset_period()`

```
void set_reset_period (  
    int x ) [inline]
```

Set the reset period.

**Parameters**

$x$	Reset period
-----	--------------

$x \leq 0$  means no reset.

Definition at line 255 of file hea.hh.

### 5.35.4.2 `set_selection_size()`

```
void set_selection_size (  
    int x ) [inline]
```

Set the selection size.

The selection size is the number of selected individuals in the population.

Definition at line 247 of file hea.hh.

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

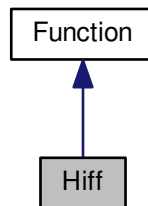
- `lib/hnco/algorithms/hea/hea.hh`

## 5.36 Hiff Class Reference

Hierarchical if and only if.

```
#include <hnco/functions/theory.hh>
```

Inheritance diagram for Hiff:



### Public Member Functions

- [Hiff](#) (int bv\_size)  
*Constructor.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- [bool has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [double get\\_maximum](#) ()  
*Get the global maximum.*

### Private Attributes

- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*
- [size\\_t \\_depth](#)  
*Tree depth.*

### 5.36.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 169 of file theory.hh.

## 5.36.2 Member Function Documentation

### 5.36.2.1 get\_maximum()

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Returns

$(i + 1) * 2^i$  where  $2^i = \_bv\_size$

Reimplemented from [Function](#).

Definition at line 195 of file theory.hh.

### 5.36.2.2 has\_known\_maximum()

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 191 of file theory.hh.

The documentation for this class was generated from the following files:

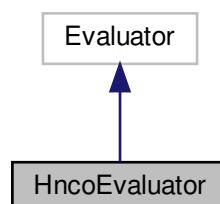
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.37 HncoEvaluator Class Reference

Evaluator for HNC0 functions.

```
#include <hnco/algorithms/eda/hnco-evaluator.hh>
```

Inheritance diagram for HncoEvaluator:



## Public Member Functions

- [HncoEvaluator](#) ([hnco::function::Function](#) \*function)  
*Constructor.*
- float [evaluate](#) (const std::vector< bool > &x)  
*Evaluate a bit vector.*

## Private Attributes

- [hnco::function::Function](#) \* [\\_function](#)  
*HNCO function.*
- [hnco::bit\\_vector\\_t](#) [\\_bv](#)  
*Argument of HNCO function.*

### 5.37.1 Detailed Description

Evaluator for HNCO functions.

Definition at line 35 of file hnco-evaluator.hh.

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

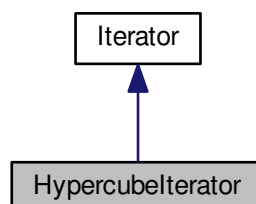
- lib/hnco/algorithms/eda/hnco-evaluator.hh

## 5.38 Hypercubeliterator Class Reference

Hypercube iterator.

```
#include <hnco/iterator.hh>
```

Inheritance diagram for Hypercubeliterator:



## Public Member Functions

- [Hypercubeliterator](#) (int n)  
*Constructor.*
- bool [has\\_next](#) ()  
*Has next bit vector.*
- const [bit\\_vector\\_t](#) & [next](#) ()  
*Next bit vector.*

## Additional Inherited Members

### 5.38.1 Detailed Description

Hypercube iterator.

Implemented as a simple binary adder.

Definition at line 69 of file iterator.hh.

The documentation for this class was generated from the following files:

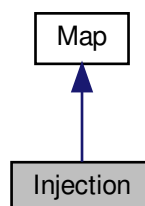
- lib/hnco/iterator.hh
- lib/hnco/iterator.cc

## 5.39 Injection Class Reference

[Injection](#).

```
#include <hnco/map.hh>
```

Inheritance diagram for Injection:



## Public Member Functions

- [Injection](#) (const std::vector< std::size\_t > &bit\_positions, std::size\_t output\_size)  
*Constructor.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- size\_t [get\\_input\\_size](#) ()  
*Get input size.*
- size\_t [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

## Private Attributes

- std::vector< std::size\_t > [\\_bit\\_positions](#)  
*Bit positions.*
- std::size\_t [\\_output\\_size](#)  
*Output size.*

### 5.39.1 Detailed Description

[Injection](#).

An injection copies the bits of input x to given positions of output y.

Let  $I = \{i_1, i_2, \dots, i_m\}$  be a subset of  $\{1, 2, \dots, n\}$ .

An injection f from  $Z_2^m$  to  $Z_2^n$ , where  $n \geq m$ , is defined by  $f(x) = y$ , where, for all  $j \in \{1, 2, \dots, m\}$ ,  $y_{i_j} = x_j$ .

If f is a projection and g is an injection with the same bit positions then their composition  $f \circ g$  is the identity.

Definition at line 396 of file map.hh.

### 5.39.2 Constructor & Destructor Documentation

#### 5.39.2.1 Injection()

```
Injection (
    const std::vector< std::size_t > & bit_positions,
    std::size_t output_size )
```

Constructor.

The input size of the map is given by the size of bit\_positions.



## Parameters

<i>bit_positions</i>	Bit positions in the output to where input bits are copied
<i>output_size</i>	Output size

## Precondition

`output_size >= bit_positions.size()`

Definition at line 144 of file map.cc.

The documentation for this class was generated from the following files:

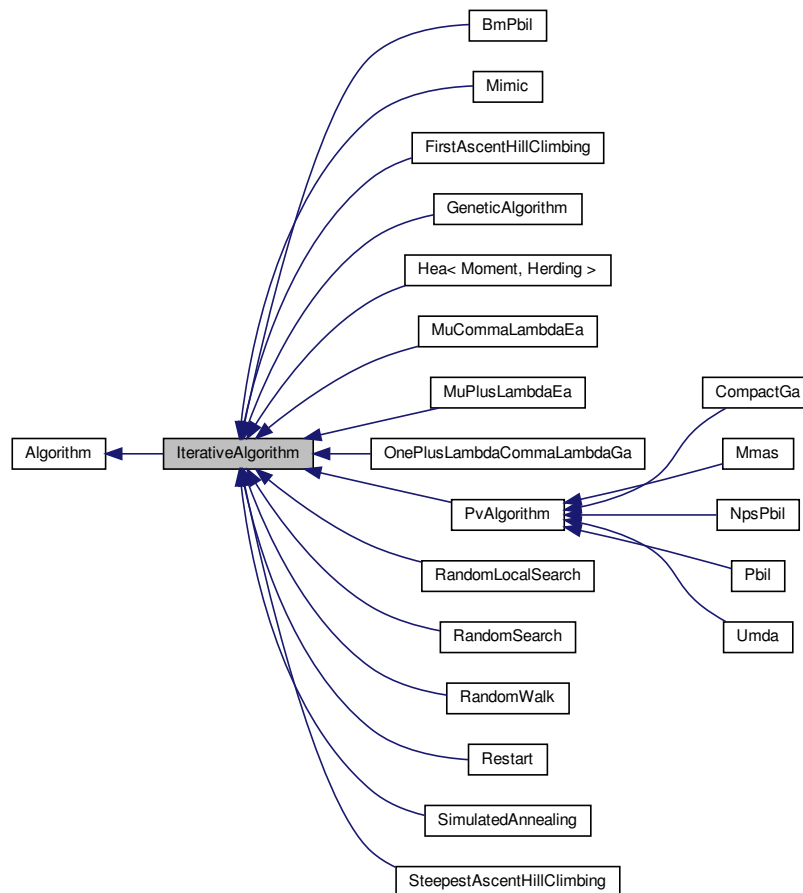
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.40 IterativeAlgorithm Class Reference

Iterative search.

```
#include <hnco/algorithms/algorithm.hh>
```

Inheritance diagram for IterativeAlgorithm:



## Public Member Functions

- [IterativeAlgorithm](#) (int n)  
*Constructor.*
- void [maximize](#) ()  
*Maximize.*

## Setters

- void [set\\_num\\_iterations](#) (int x)  
*Set the number of iterations.*

## Protected Member Functions

- virtual void [iterate](#) ()=0  
*Single iteration.*
- virtual void [log](#) ()  
*Log.*

## Protected Attributes

- int [\\_iteration](#)  
*Current iteration.*
- bool [\\_something\\_to\\_log](#)  
*Something to log.*

## Parameters

- int [\\_num\\_iterations](#) = 0  
*Number of iterations.*

### 5.40.1 Detailed Description

Iterative search.

Definition at line 169 of file algorithm.hh.

### 5.40.2 Constructor & Destructor Documentation

#### 5.40.2.1 IterativeAlgorithm()

```
IterativeAlgorithm (  
    int n ) [inline]
```

Constructor.

## Parameters

$n$	Size of bit vectors
-----	---------------------

Definition at line 199 of file algorithm.hh.

### 5.40.3 Member Function Documentation

#### 5.40.3.1 maximize()

```
void maximize ( ) [virtual]
```

Maximize.

Inside the loop:

- call [iterate\(\)](#)
- call [log\(\)](#)

## Warning

If an exception such as LocalMaximum is thrown by [iterate\(\)](#), [log\(\)](#) will not be called. However, hnco reports the maximum at the end of the search.

Implements [Algorithm](#).

Definition at line 77 of file algorithm.cc.

#### 5.40.3.2 set\_num\_iterations()

```
void set_num_iterations (
    int x ) [inline]
```

Set the number of iterations.

## Parameters

$x$	Number of iterations
-----	----------------------

$x \leq 0$  means indefinite

Definition at line 223 of file algorithm.hh.

The documentation for this class was generated from the following files:

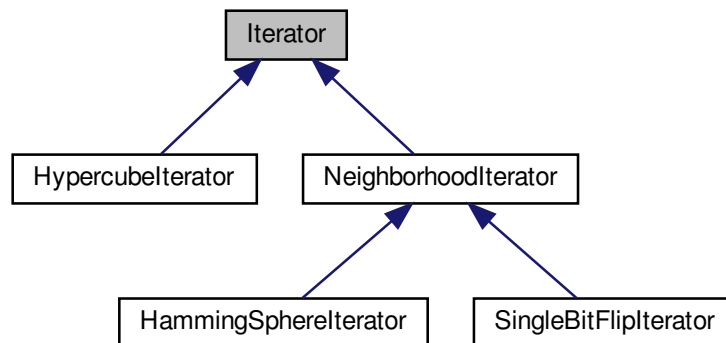
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

## 5.41 Iterator Class Reference

[Iterator](#) over bit vectors.

```
#include <hnco/iterator.hh>
```

Inheritance diagram for Iterator:



### Public Member Functions

- [Iterator](#) (int n)  
*Constructor.*
- virtual [~Iterator](#) ()  
*Destructor.*
- virtual void [init](#) ()  
*Initialization.*
- virtual bool [has\\_next](#) ()=0  
*Has next bit vector.*
- virtual const [bit\\_vector\\_t](#) & [next](#) ()=0  
*Next bit vector.*

### Protected Attributes

- [bit\\_vector\\_t](#) [\\_current](#)  
*Current bit vector.*
- bool [\\_initial\\_state](#) = true  
*Flag for initial state.*

### 5.41.1 Detailed Description

[Iterator](#) over bit vectors.

Definition at line 34 of file iterator.hh.

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

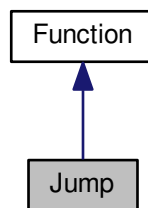
- lib/hnco/iterator.hh

## 5.42 Jump Class Reference

[Jump](#).

```
#include <hnco/functions/jump.hh>
```

Inheritance diagram for Jump:



### Public Member Functions

- [Jump](#) (int bv\_size, int gap)  
*Constructor.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*

### Private Attributes

- size\_t [\\_bv\\_size](#)  
*Bit vector size.*
- int [\\_gap](#)  
*Gap.*

### 5.42.1 Detailed Description

[Jump](#).

Reference:

H. Mühlenbein and T. Mahnig. 2001. Evolutionary Algorithms: From Recombination to Search Distributions. In Theoretical Aspects of Evolutionary Computing, Leila Kallel, Bart Naudts, and Alex Rogers (Eds.). Springer Berlin Heidelberg, 135–174.

Definition at line 40 of file jump.hh.

### 5.42.2 Member Function Documentation

#### 5.42.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

Returns

`_bv_size`

Reimplemented from [Function](#).

Definition at line 66 of file jump.hh.

#### 5.42.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

Returns

`true`

Reimplemented from [Function](#).

Definition at line 62 of file jump.hh.

The documentation for this class was generated from the following files:

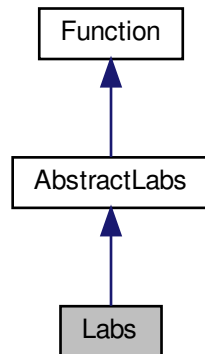
- lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

## 5.43 Labs Class Reference

Low autocorrelation binary sequences.

```
#include <hnco/functions/labs.hh>
```

Inheritance diagram for Labs:



### Public Member Functions

- [Labs](#) (int n)  
*Constructor.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Additional Inherited Members

#### 5.43.1 Detailed Description

Low autocorrelation binary sequences.

Reference:

S Mertens. 1996. Exhaustive search for low-autocorrelation binary sequences. Journal of Physics A: Mathematical and General 29, 18 (1996), L473.

<http://stacks.iop.org/0305-4470/29/i=18/a=005>

Definition at line 64 of file labs.hh.

The documentation for this class was generated from the following files:

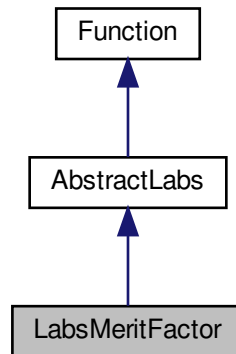
- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

## 5.44 LabsMeritFactor Class Reference

Low autocorrelation binary sequences merit factor.

```
#include <hnco/functions/labs.hh>
```

Inheritance diagram for LabsMeritFactor:



### Public Member Functions

- [LabsMeritFactor](#) (int n)  
*Constructor.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Additional Inherited Members

#### 5.44.1 Detailed Description

Low autocorrelation binary sequences merit factor.

Reference:

S Mertens. 1996. Exhaustive search for low-autocorrelation binary sequences. Journal of Physics A: Mathematical and General 29, 18 (1996), L473.

<http://stacks.iop.org/0305-4470/29/i=18/a=005>

Definition at line 89 of file labs.hh.

The documentation for this class was generated from the following files:

- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

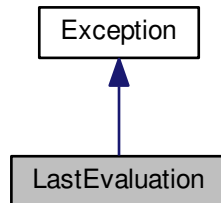


## 5.45 LastEvaluation Class Reference

Last evaluation.

```
#include <hnco/exception.hh>
```

Inheritance diagram for LastEvaluation:



### 5.45.1 Detailed Description

Last evaluation.

Definition at line 79 of file exception.hh.

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

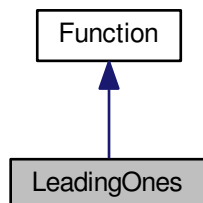
- lib/hnco/exception.hh

## 5.46 LeadingOnes Class Reference

Leading ones.

```
#include <hnco/functions/theory.hh>
```

Inheritance diagram for LeadingOnes:



## Public Member Functions

- [LeadingOnes](#) (int bv\_size)  
*Constructor.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- [bool has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [double get\\_maximum](#) ()  
*Get the global maximum.*

## Private Attributes

- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*

### 5.46.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 97 of file theory.hh.

### 5.46.2 Member Function Documentation

#### 5.46.2.1 [get\\_maximum\(\)](#)

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

Returns

[\\_bv\\_size](#)

Reimplemented from [Function](#).

Definition at line 121 of file theory.hh.

### 5.46.2.2 has\_known\_maximum()

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 117 of file theory.hh.

The documentation for this class was generated from the following files:

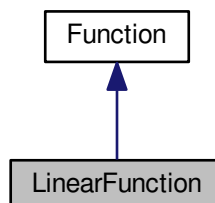
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.47 LinearFunction Class Reference

Linear function.

```
#include <hnco/functions/linear-function.hh>
```

Inheritance diagram for LinearFunction:



### Public Member Functions

- [LinearFunction](#) ()  
*Constructor.*
- void [random](#) (int n)  
*Random instance.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*

## Private Member Functions

- `template<class Archive >`  
`void serialize (Archive &ar, const unsigned int version)`  
*Serialize.*

## Private Attributes

- `std::vector< double > \_weights`  
*Weights.*

## Friends

- class **`boost::serialization::access`**

### 5.47.1 Detailed Description

Linear function.

Definition at line 40 of file linear-function.hh.

### 5.47.2 Member Function Documentation

#### 5.47.2.1 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

`true`

Reimplemented from [Function](#).

Definition at line 76 of file linear-function.hh.

#### 5.47.2.2 `random()`

```
void random (
    int n )
```

Random instance.

## Parameters

$n$	Size of bit vectors
-----	---------------------

Definition at line 33 of file linear-function.cc.

The documentation for this class was generated from the following files:

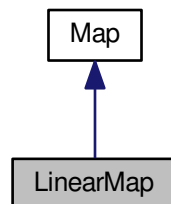
- lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

## 5.48 LinearMap Class Reference

Linear map.

```
#include <hnco/map.hh>
```

Inheritance diagram for LinearMap:



### Public Member Functions

- void [random](#) (int rows, int cols, bool surjective)  
*Random instance.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- [size\\_t](#) [get\\_input\\_size](#) ()  
*Get input size.*
- [size\\_t](#) [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

## Private Member Functions

- `template<class Archive >`  
void [save](#) (Archive &ar, const unsigned int version) const  
*Save.*
- `template<class Archive >`  
void [load](#) (Archive &ar, const unsigned int version)  
*Load.*

## Private Attributes

- [bit\\_matrix\\_t\\_bm](#)  
*Bit matrix.*

## Friends

- class **boost::serialization::access**

### 5.48.1 Detailed Description

Linear map.

A linear map  $f$  from  $Z_2^m$  to  $Z_2^n$  is defined by  $f(x) = Ax$ , where  $A$  is an  $n \times m$  bit matrix.

Definition at line 193 of file map.hh.

### 5.48.2 Member Function Documentation

#### 5.48.2.1 is\_surjective()

```
bool is_surjective ( ) [virtual]
```

Check for surjective map.

#### Returns

true if `rank(_bm) == bm_num_rows(_bm)`

Reimplemented from [Map](#).

Definition at line 90 of file map.cc.

#### 5.48.2.2 random()

```
void random (
    int rows,
    int cols,
    bool surjective )
```

Random instance.

## Parameters

<i>rows</i>	Number of rows
<i>cols</i>	Number of columns
<i>surjective</i>	Flag to ensure a surjective map

## Exceptions

<i>Error</i>	
--------------	--

Definition at line 61 of file map.cc.

The documentation for this class was generated from the following files:

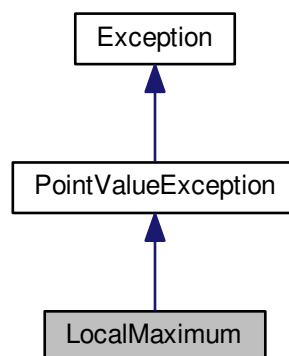
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.49 LocalMaximum Class Reference

Local maximum.

```
#include <hnco/exception.hh>
```

Inheritance diagram for LocalMaximum:



## Public Member Functions

- [LocalMaximum](#) (const [point\\_value\\_t](#) &pv)  
*Const.*

## Additional Inherited Members

### 5.49.1 Detailed Description

Local maximum.

Definition at line 70 of file exception.hh.

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

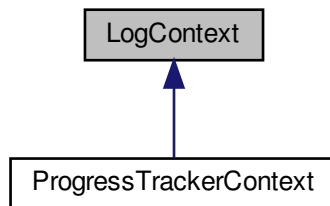
- lib/hnco/exception.hh

## 5.50 LogContext Class Reference

Log context.

```
#include <hnco/algorithms/log-context.hh>
```

Inheritance diagram for LogContext:



## Public Member Functions

- virtual std::string [get\\_context](#) ()=0  
*Get context.*

### 5.50.1 Detailed Description

Log context.

A log context gives an algorithm more information about what is going on during optimization than what can be gained through its function. In particular, its function may not be a function controller. Information is provided through a log context in the form of a string.

Definition at line 39 of file log-context.hh.

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

- lib/hnco/algorithms/log-context.hh

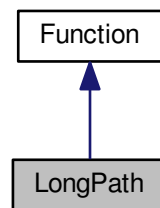


## 5.51 LongPath Class Reference

Long path.

```
#include <hnco/functions/long-path.hh>
```

Inheritance diagram for LongPath:



### Public Member Functions

- [LongPath](#) (int bv\_size, int prefix\_length)

*Constructor.*

- double [eval](#) (const [bit\\_vector\\_t](#) &)

*Evaluate a bit vector.*

### Information about the function

- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*

### Private Attributes

- size\_t [\\_bv\\_size](#)  
*Bit vector size.*
- int [\\_prefix\\_length](#)  
*Prefix length.*

### 5.51.1 Detailed Description

Long path.

Long paths have been introduced by Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Here we mostly follow the definition given by Thomas Jansen (see references below).

As an example, here is the 2-long path of dimension 4:

- 0000
- 0001
- 0011
- 0111
- 1111
- 1101
- 1100

The fitness is increasing along the path. The fitness on the complementary of the path is defined as a linear function pointing to the beginning of the path.

To help with the detection of maximum, we have dropped the constant  $n^2$  whose sole purpose was to make the function non negative.

References:

Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb, "Long Path Problems", PPSN III, 1994.

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 62 of file long-path.hh.

### 5.51.2 Member Function Documentation

#### 5.51.2.1 get\_maximum()

```
double get_maximum ( ) [virtual]
```

Get the global maximum.

Let  $n$  be the bit vector size and  $k$  the prefix length which must divide  $n$ . Then the maximum is  $k2^{n/k} - k + 1$ .

Exceptions

Error	
-------	--

Reimplemented from [Function](#).

Definition at line 62 of file long-path.cc.

#### 5.51.2.2 has\_known\_maximum()

```
bool has_known_maximum ( ) [virtual]
```

Check for a known maximum.

Let  $n$  be the bit vector size and  $k$  the prefix length which must divide  $n$ .

We have to check that the maximum can be represented exactly as a double, that is, it must be lower or equal to  $2^{53}$ . We are a little bit more conservative with the following test.

If  $\log_2(k) + n/k \leq 53$  then returns true else returns false.

Reimplemented from [Function](#).

Definition at line 52 of file long-path.cc.

The documentation for this class was generated from the following files:

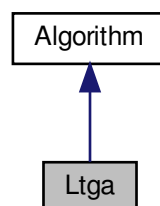
- lib/hnco/functions/long-path.hh
- lib/hnco/functions/long-path.cc

## 5.52 Ltga Class Reference

Linkage Tree Genetic Algorithm.

```
#include <hnco/algorithms/eda/ltga.hh>
```

Inheritance diagram for Ltga:



## Public Member Functions

- [Ltga](#) (int n)  
*Constructor.*
- void [maximize](#) ()  
*Maximize.*
- void [set\\_population\\_size](#) (int n)  
*Set population size.*

## Private Attributes

- int [\\_population\\_size](#) = 10  
*Population size.*

## Additional Inherited Members

### 5.52.1 Detailed Description

Linkage Tree Genetic Algorithm.

Implementation of the Linkage Tree Genetic Algorithm Designed to match the variant in the paper: "Hierarchical problem solving with the linkage tree genetic algorithm" by D. Thierens and P. A. N. Bosman

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 40 of file Ltga.hh.

The documentation for this class was generated from the following files:

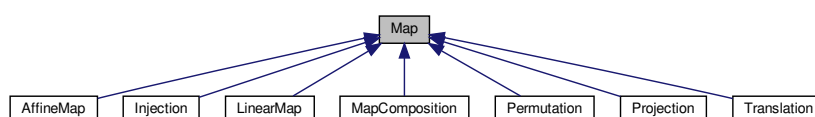
- lib/hnco/algorithms/eda/Ltga.hh
- lib/hnco/algorithms/eda/Ltga.cc

## 5.53 Map Class Reference

[Map](#).

```
#include <hnco/map.hh>
```

Inheritance diagram for Map:



## Public Member Functions

- virtual [~Map](#) ()  
*Destructor.*
- virtual void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)=0  
*Map.*
- virtual size\_t [get\\_input\\_size](#) ()=0  
*Get input size.*
- virtual size\_t [get\\_output\\_size](#) ()=0  
*Get output size.*
- virtual bool [is\\_surjective](#) ()  
*Check for surjective map.*

### 5.53.1 Detailed Description

[Map](#).

Definition at line 39 of file map.hh.

### 5.53.2 Member Function Documentation

#### 5.53.2.1 [is\\_surjective\(\)](#)

```
virtual bool is_surjective ( ) [inline], [virtual]
```

Check for surjective map.

#### Returns

false

Reimplemented in [Projection](#), [Injection](#), [MapComposition](#), [AffineMap](#), [LinearMap](#), [Permutation](#), and [Translation](#).

Definition at line 59 of file map.hh.

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

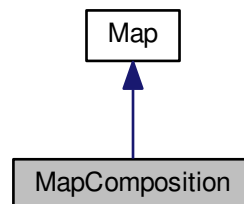
- lib/hnco/map.hh

## 5.54 MapComposition Class Reference

[Map](#) composition.

```
#include <hnco/map.hh>
```

Inheritance diagram for MapComposition:



### Public Member Functions

- [MapComposition](#) ()  
*Default constructor.*
- [MapComposition](#) ([Map](#) \*outer, [Map](#) \*inner)  
*Constructor.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- size\_t [get\\_input\\_size](#) ()  
*Get input size.*
- size\_t [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

### Private Attributes

- [Map](#) \* [\\_outer](#)  
*Outer map.*
- [Map](#) \* [\\_inner](#)  
*Inner map.*
- [bit\\_vector\\_t](#) [\\_bv](#)  
*Temporary bit vector.*

### 5.54.1 Detailed Description

[Map](#) composition.

The resulting composition  $f$  is defined for all bit vector  $x$  by  $f(x) = \text{outer}(\text{inner}(x))$ .

Definition at line 327 of file map.hh.

## 5.54.2 Constructor & Destructor Documentation

### 5.54.2.1 MapComposition()

```
MapComposition (
    Map * outer,
    Map * inner ) [inline]
```

Constructor.

#### Parameters

<i>outer</i>	outer map
<i>inner</i>	inner map

#### Precondition

```
outer->get_input_size() == inner->get_output_size()
```

Definition at line 351 of file map.hh.

## 5.54.3 Member Function Documentation

### 5.54.3.1 is\_surjective()

```
bool is_surjective ( ) [inline], [virtual]
```

Check for surjective map.

#### Returns

true if both maps are surjective

Reimplemented from [Map](#).

Definition at line 375 of file map.hh.

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

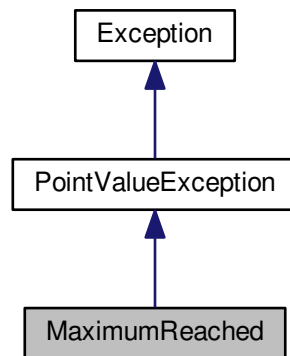
- lib/hnco/map.hh

## 5.55 MaximumReached Class Reference

Maximum reached.

```
#include <hnco/exception.hh>
```

Inheritance diagram for MaximumReached:



### Public Member Functions

- [MaximumReached](#) (const [point\\_value\\_t](#) &pv)  
*Constructor.*

### Additional Inherited Members

#### 5.55.1 Detailed Description

Maximum reached.

Definition at line 52 of file exception.hh.

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

- lib/hnco/exception.hh

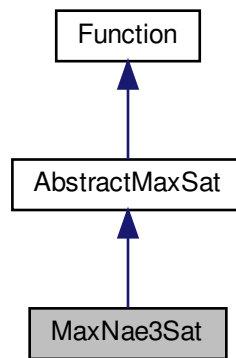


## 5.56 MaxNae3Sat Class Reference

Max not-all-equal 3SAT.

```
#include <hnco/functions/max-sat.hh>
```

Inheritance diagram for MaxNae3Sat:



### Public Member Functions

- [MaxNae3Sat](#) ()  
*Default constructor.*
- void [load](#) (std::istream &stream)  
*Load an instance.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Additional Inherited Members

#### 5.56.1 Detailed Description

Max not-all-equal 3SAT.

Reference:

Christos M. Papadimitriou. 1994. Computational complexity. Addison-Wesley, Reading, Massachusetts.

Definition at line 125 of file max-sat.hh.

#### 5.56.2 Member Function Documentation

##### 5.56.2.1 load()

```
void load (
    std::istream & stream ) [virtual]
```

Load an instance.

### Exceptions

Error	
-------	--

Reimplemented from [AbstractMaxSat](#).

Definition at line 282 of file max-sat.cc.

The documentation for this class was generated from the following files:

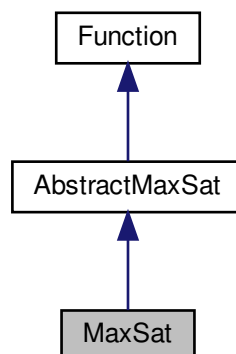
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

## 5.57 MaxSat Class Reference

MAX-SAT.

```
#include <hnco/functions/max-sat.hh>
```

Inheritance diagram for MaxSat:



### Public Member Functions

- [MaxSat](#) ()  
*Default constructor.*
- void [random](#) (int n, int k, int c)  
*Random instance.*
- void [random](#) (const [bit\\_vector\\_t](#) &solution, int k, int c)  
*Random instance with satisfiable expression.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Additional Inherited Members

### 5.57.1 Detailed Description

MAX-SAT.

Reference:

Christos M. Papadimitriou. 1994. Computational complexity. Addison-Wesley, Reading, Massachusetts.

Definition at line 81 of file max-sat.hh.

### 5.57.2 Member Function Documentation

#### 5.57.2.1 `random()` [1/2]

```
void random (
    int n,
    int k,
    int c )
```

Random instance.

Parameters

<i>n</i>	Size of bit vectors
<i>k</i>	Number of literals per clause
<i>c</i>	Number of clauses

Definition at line 190 of file max-sat.cc.

#### 5.57.2.2 `random()` [2/2]

```
void random (
    const bit_vector_t & solution,
    int k,
    int c )
```

Random instance with satisfiable expression.

Warning

Since the expression is satisfiable, the maximum of the function is equal to the number of clauses in the expression. However, this information is lost in the save and load cycle as the archive format only manages the expression itself.

**Parameters**

<i>solution</i>	Solution
<i>k</i>	Number of literals per clause
<i>c</i>	Number of clauses

Definition at line 218 of file max-sat.cc.

The documentation for this class was generated from the following files:

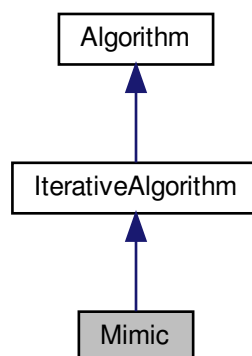
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

## 5.58 Mimic Class Reference

Mutual information maximizing input clustering.

```
#include <hnco/algorithms/eda/mimic.hh>
```

Inheritance diagram for Mimic:

**Public Member Functions**

- [Mimic](#) (int n, int population\_size)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

**Setters**

- void [set\\_selection\\_size](#) (int x)  
*Set the selection size.*

## Protected Member Functions

- void `iterate` ()  
*Single iteration.*
- void `sample` (`bit_vector_t` &bv)  
*Sample a bit vector.*
- void `compute_conditional_entropy` (`std::size_t` index)  
*Compute conditional entropy.*
- void `update_model` ()  
*Update model.*

## Protected Attributes

- `Population` `_population`  
*Population.*
- `permutation_t` `_permutation`  
*Permutation.*
- `std::array< pv_t, 2 >` `_parameters`  
*Model parameters.*
- `pv_t` `_mean`  
*Mean of selected bit vectors.*
- `std::vector< double >` `_entropies`  
*Conditional entropies.*
- `std::array< std::array< int, 2 >, 2 >` `_table`  
*Contingency table.*
- `double` `_lower_bound`  
*Lower bound of probability.*
- `double` `_upper_bound`  
*Upper bound of probability.*

## Parameters

- `int` `_selection_size`  
*Selection size.*

### 5.58.1 Detailed Description

Mutual information maximizing input clustering.

This implementation differs from the algorithm described in the reference below in that it constrains all probabilities (marginal and conditional) to stay away from the values 0 and 1 by a fixed margin equal to  $1/n$ , as usually done in algorithms such as [Pbil](#) or [Umda](#).

Reference:

Jeremy S. De Bonet and Charles L. Isbell and Jr. and Paul Viola, MIMIC: Finding Optima by Estimating Probability Densities, in Advances in Neural Information Processing Systems, 1996, MIT Press.

Definition at line 52 of file `mimic.hh`.

The documentation for this class was generated from the following files:

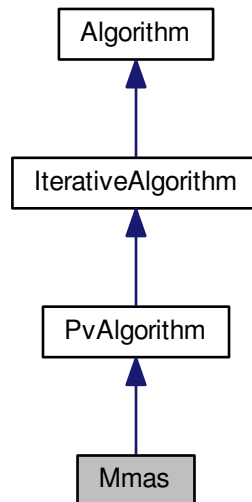
- `lib/hnco/algorithms/eda/mimic.hh`
- `lib/hnco/algorithms/eda/mimic.cc`

## 5.59 Mmas Class Reference

Max-min ant system.

```
#include <hnco/algorithms/pv/mmas.hh>
```

Inheritance diagram for Mmas:



### Public Member Functions

- [Mmas](#) (int n)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_compare](#) (std::function< bool(double, double)> x)  
*Set the binary operator for comparing evaluations.*
- void [set\\_learning\\_rate](#) (double x)  
*Set the learning rate.*

### Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*

## Protected Attributes

- [bit\\_vector\\_t\\_x](#)  
*Candidate solution.*

## Parameters

- `std::function< bool(double, double)> \_compare = std::greater_equal<double>()`  
*Binary operator for comparing evaluations.*
- `double \_learning\_rate = 1e-3`  
*Learning rate.*

## Additional Inherited Members

### 5.59.1 Detailed Description

Max-min ant system.

Reference:

Thomas Stützle and Holger H. Hoos. 2000. MAX-MIN Ant System. *Future Generation Computer Systems* 16, 8 (2000), 889–914.

Definition at line 41 of file `mmas.hh`.

The documentation for this class was generated from the following files:

- `lib/hnco/algorithms/pv/mmas.hh`
- `lib/hnco/algorithms/pv/mmas.cc`

## 5.60 Model Class Reference

[Model](#) of a Boltzmann machine.

```
#include <hnco/algorithms/bm-pbil/model.hh>
```

## Public Member Functions

- [Model](#) (int n)  
*Constructor.*
- void [init](#) ()  
*Initialize.*
- void [reset\\_mc](#) ()  
*Reset Markov chain.*
- void [gibbs\\_sampler](#) (size\_t i)  
*A Gibbs sampler cycle.*
- void [gibbs\\_sampler\\_synchronous](#) ()  
*A synchronous Gibbs sampler.*
- const [bit\\_vector\\_t](#) & [get\\_state](#) ()  
*Get the state of the Gibbs sampler.*
- void [update](#) (const [ModelParameters](#) &p, const [ModelParameters](#) &q, double rate)  
*Update parameters in the direction of p and away from q.*
- double [norm\\_infinite](#) ()  
*Infinite norm of the parameters.*
- double [norm\\_l1](#) ()  
*l1 norm of the parameters*

## Private Attributes

- [ModelParameters \\_model\\_parameters](#)  
*Model parameters.*
- [bit\\_vector\\_t \\_state](#)  
*State of the Gibbs sampler.*
- [pv\\_t \\_pv](#)  
*Probability vector for synchronous Gibbs sampling.*

### 5.60.1 Detailed Description

[Model](#) of a Boltzmann machine.

Definition at line 75 of file `model.hh`.

The documentation for this class was generated from the following files:

- `lib/hnco/algorithms/bm-pbil/model.hh`
- `lib/hnco/algorithms/bm-pbil/model.cc`

## 5.61 ModelParameters Class Reference

Parameters of a Boltzmann machine.

```
#include <hnco/algorithms/bm-pbil/model.hh>
```

## Public Member Functions

- [ModelParameters](#) (int n)  
*Constructor.*
- void [init](#) ()  
*Initialize.*
- void [add](#) (const [bit\\_vector\\_t](#) &x)  
*Add a bit\_vector\_t.*
- void [average](#) (int count)  
*Compute averages.*
- void [update](#) (const [ModelParameters](#) &p, const [ModelParameters](#) &q, double rate)  
*Update parameters in the direction of p and away from q.*
- double [norm\\_infinite](#) ()  
*Infinite norm of the parameters.*
- double [norm\\_l1](#) ()  
*l1 norm of the parameters*

## Private Attributes

- `std::vector< std::vector< double > >` [\\_weight](#)  
*Weights.*
- `std::vector< double >` [\\_bias](#)  
*Bias.*



## Friends

- class **Model**

### 5.61.1 Detailed Description

Parameters of a Boltzmann machine.

Definition at line 36 of file model.hh.

The documentation for this class was generated from the following files:

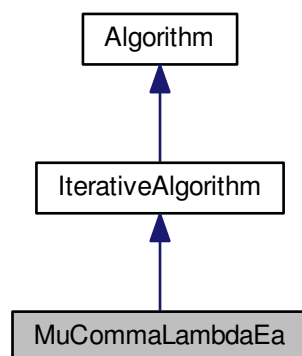
- lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

## 5.62 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

```
#include <hnco/algorithms/ea/mu-comma-lambda-ea.hh>
```

Inheritance diagram for MuCommaLambdaEa:



## Public Member Functions

- [MuCommaLambdaEa](#) (int n, int mu, int lambda)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

## Setters

- void [set\\_mutation\\_probability](#) (double x)  
*Set the mutation probability.*
- void [set\\_allow\\_stay](#) (bool x)  
*Set the flag \_allow\_stay.*

## Private Member Functions

- void `iterate` ()  
*Single iteration.*

## Private Attributes

- `Population _parents`  
*Parents.*
- `Population _offsprings`  
*Offsprings.*
- `neighborhood::BernoulliProcess _mutation`  
*Mutation operator.*
- `std::uniform_int_distribution< int > _select_parent`  
*Select parent.*

## Parameters

- double `_mutation_probability`  
*Mutation probability.*
- bool `_allow_stay` = false  
*Allow stay.*

## Additional Inherited Members

### 5.62.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 41 of file mu-comma-lambda-ea.hh.

### 5.62.2 Constructor & Destructor Documentation

#### 5.62.2.1 MuCommaLambdaEa()

```
MuCommaLambdaEa (
    int n,
    int mu,
    int lambda ) [inline]
```

Constructor.

## Parameters

<i>n</i>	Size of bit vectors
<i>mu</i>	Parent population size
<i>lambda</i>	Offspring population size

Definition at line 79 of file mu-comma-lambda-ea.hh.

### 5.62.3 Member Function Documentation

#### 5.62.3.1 set\_allow\_stay()

```
void set_allow_stay (
    bool x ) [inline]
```

Set the flag `_allow_stay`.

In case no mutation occurs allow the current bit vector to stay unchanged.

Definition at line 102 of file mu-comma-lambda-ea.hh.

The documentation for this class was generated from the following files:

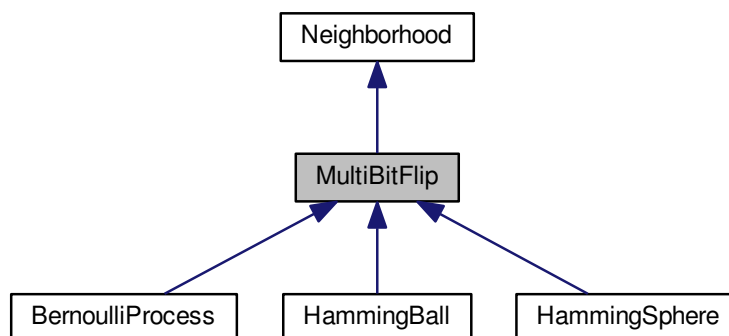
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

## 5.63 MultiBitFlip Class Reference

Multi bit flip.

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for MultiBitFlip:



## Public Member Functions

- [MultiBitFlip](#) (int n)  
*Constructor.*

## Protected Member Functions

- void [bernoulli\\_trials](#) (int k)  
*Sample a given number of bits using Bernoulli trials.*
- void [reservoir\\_sampling](#) (int k)  
*Sample a given number of bits using resevoir sampling.*

## Additional Inherited Members

### 5.63.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

### 5.63.2 Constructor & Destructor Documentation

#### 5.63.2.1 MultiBitFlip()

```
MultiBitFlip (  
    int n ) [inline]
```

Constructor.

#### Parameters

<i>n</i>	Size of bit vectors
----------	---------------------

Definition at line 206 of file neighborhood.hh.

### 5.63.3 Member Function Documentation

#### 5.63.3.1 bernoulli\_trials()

```
void bernoulli_trials (  
    int k ) [protected]
```

Sample a given number of bits using Bernoulli trials.

## Parameters

$k$	Number of bits to sample
-----	--------------------------

Definition at line 34 of file neighborhood.cc.

### 5.63.3.2 reservoir\_sampling()

```
void reservoir_sampling (  
    int  $k$  )    [protected]
```

Sample a given number of bits using resevoir sampling.

## Parameters

$k$	Number of bits to sample
-----	--------------------------

Definition at line 52 of file neighborhood.cc.

The documentation for this class was generated from the following files:

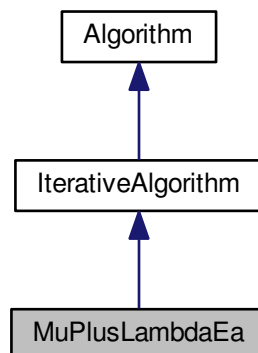
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.64 MuPlusLambdaEa Class Reference

(mu+lambda) EA.

```
#include <hnco/algorithms/ea/mu-plus-lambda-ea.hh>
```

Inheritance diagram for MuPlusLambdaEa:



## Public Member Functions

- [MuPlusLambdaEa](#) (int n, int mu, int lambda)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

## Setters

- void [set\\_mutation\\_probability](#) (double x)  
*Set the mutation probability.*
- void [set\\_allow\\_stay](#) (bool x)  
*Set the flag \_allow\_stay.*

## Private Member Functions

- void [iterate](#) ()  
*Single iteration.*

## Private Attributes

- [Population \\_parents](#)  
*Parents.*
- [Population \\_offsprings](#)  
*Offsprings.*
- [neighborhood::BernoulliProcess \\_mutation](#)  
*Mutation operator.*
- [std::uniform\\_int\\_distribution< int > \\_select\\_parent](#)  
*Select parent.*

## Parameters

- double [\\_mutation\\_probability](#)  
*Mutation probability.*
- bool [\\_allow\\_stay](#) = false  
*Allow stay.*

## Additional Inherited Members

### 5.64.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 40 of file mu-plus-lambda-ea.hh.

## 5.64.2 Constructor & Destructor Documentation

### 5.64.2.1 MuPlusLambdaEa()

```
MuPlusLambdaEa (
    int n,
    int mu,
    int lambda ) [inline]
```

Constructor.

#### Parameters

<i>n</i>	Size of bit vectors
<i>mu</i>	Parent population size
<i>lambda</i>	Offspring population size

Definition at line 78 of file mu-plus-lambda-ea.hh.

## 5.64.3 Member Function Documentation

### 5.64.3.1 set\_allow\_stay()

```
void set_allow_stay (
    bool x ) [inline]
```

Set the flag `_allow_stay`.

In case no mutation occurs allow the current bit vector to stay unchanged.

Definition at line 101 of file mu-plus-lambda-ea.hh.

The documentation for this class was generated from the following files:

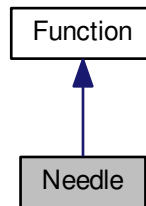
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

## 5.65 Needle Class Reference

[Needle](#) in a haystack.

```
#include <hnco/functions/theory.hh>
```

Inheritance diagram for Needle:



### Public Member Functions

- [Needle](#) (int bv\_size)  
*Constructor.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*

### Private Attributes

- size\_t [\\_bv\\_size](#)  
*Bit vector size.*

#### 5.65.1 Detailed Description

[Needle](#) in a haystack.

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 133 of file theory.hh.



## 5.65.2 Member Function Documentation

### 5.65.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Returns

1

Reimplemented from [Function](#).

Definition at line 157 of file theory.hh.

### 5.65.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 153 of file theory.hh.

The documentation for this class was generated from the following files:

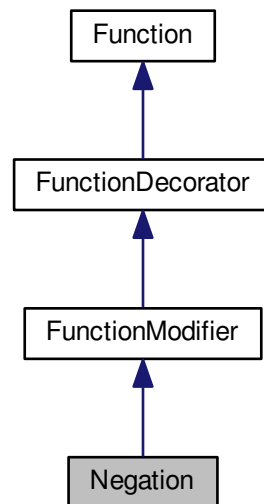
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.66 Negation Class Reference

[Negation](#).

```
#include <hnco/functions/decorators/function-modifier.hh>
```

Inheritance diagram for Negation:



### Public Member Functions

- [Negation](#) ([Function](#) \*function)  
*Constructor.*

### Information about the function

- [size\\_t](#) [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double](#) [get\\_maximum](#) ()  
*Get the global maximum.*
- [bool](#) [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [bool](#) [provides\\_incremental\\_evaluation](#) ()  
*Check whether the function provides incremental evaluation.*

### Evaluation

- [double](#) [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- [double](#) [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped↔  
\_bits)  
*Incremental evaluation.*

## Additional Inherited Members

### 5.66.1 Detailed Description

[Negation](#).

Use cases:

- for algorithms which minimize rather than maximize a function
- for functions one wishes to minimize
- when minimization is needed inside an algorithm

Definition at line 58 of file function-modifier.hh.

### 5.66.2 Member Function Documentation

#### 5.66.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Exceptions

<i>Error</i>	
--------------	--

Reimplemented from [Function](#).

Definition at line 76 of file function-modifier.hh.

#### 5.66.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

false

Reimplemented from [Function](#).

Definition at line 80 of file function-modifier.hh.

### 5.66.2.3 provides\_incremental\_evaluation()

```
bool provides_incremental_evaluation ( ) [inline], [virtual]
```

Check whether the function provides incremental evaluation.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 85 of file function-modifier.hh.

The documentation for this class was generated from the following files:

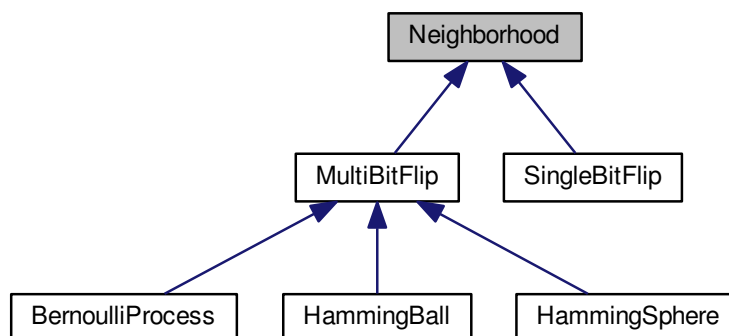
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

## 5.67 Neighborhood Class Reference

[Neighborhood](#).

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for Neighborhood:



## Public Member Functions

- [Neighborhood](#) (int n)  
*Constructor.*
- virtual [~Neighborhood](#) ()  
*Destructor.*
- virtual void [set\\_origin](#) (const [bit\\_vector\\_t](#) &x)  
*Set the origin.*
- virtual const [bit\\_vector\\_t](#) & [get\\_origin](#) ()  
*Get the origin.*
- virtual const [bit\\_vector\\_t](#) & [get\\_candidate](#) ()  
*Get the candidate bit vector.*
- virtual const [sparse\\_bit\\_vector\\_t](#) & [get\\_flipped\\_bits](#) ()  
*Get flipped bits.*
- virtual void [propose](#) ()  
*Propose a candidate bit vector.*
- virtual void [keep](#) ()  
*Keep the candidate bit vector.*
- virtual void [forget](#) ()  
*Forget the candidate bit vector.*
- virtual void [mutate](#) ([bit\\_vector\\_t](#) &bv)  
*Mutate.*
- virtual void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*

## Protected Member Functions

- virtual void [sample\\_bits](#) ()=0  
*Sample bits.*

## Protected Attributes

- [bit\\_vector\\_t \\_origin](#)  
*Origin of the neighborhood.*
- [bit\\_vector\\_t \\_candidate](#)  
*candidate bit vector*
- [std::uniform\\_int\\_distribution< int > \\_uniform\\_index\\_dist](#)  
*Uniform index distribution.*
- [sparse\\_bit\\_vector\\_t \\_flipped\\_bits](#)  
*Flipped bits.*

### 5.67.1 Detailed Description

#### Neighborhood.

A neighborhood maintains two points, `_origin` and `_candidate`. They are initialized in the same state by `set_origin`. A `Neighborhood` class must implement the member function `sample_bits` which samples the bits to flip in `_origin` to get a `_candidate`. The following member functions take care of the modifications:

- `propose`: flip `_candidate`
- `keep`: flip `_origin`
- `forget`: flip `_candidate`

After `keep` or `forget`, `_origin` and `_candidate` are in the same state again.

A `Neighborhood` class can also behave as a mutation operator through the member functions `mutate` and `map`.

Definition at line 61 of file `neighborhood.hh`.

### 5.67.2 Constructor & Destructor Documentation

#### 5.67.2.1 Neighborhood()

```
Neighborhood (
    int n ) [inline]
```

Constructor.

##### Parameters

<code>n</code>	Size of bit vectors
----------------	---------------------

Definition at line 86 of file `neighborhood.hh`.

### 5.67.3 Member Function Documentation

#### 5.67.3.1 map()

```
virtual void map (
    const bit_vector_t & input,
    bit_vector_t & output ) [inline], [virtual]
```

Map.

The output bit vector is a mutated version of the input bit vector.

## Parameters

<i>input</i>	Input bit vector
<i>output</i>	Output bit vector

Definition at line 148 of file neighborhood.hh.

## 5.67.3.2 mutate()

```
virtual void mutate (
    bit_vector_t & bv ) [inline], [virtual]
```

Mutate.

In-place mutation of the bit vector.

## Parameters

<i>bv</i>	Bit vector to mutate
-----------	----------------------

Definition at line 134 of file neighborhood.hh.

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

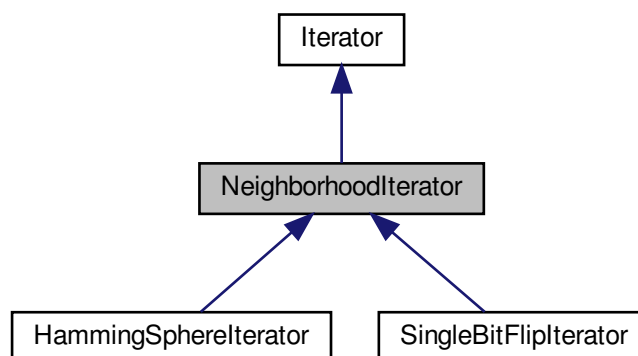
- lib/hnco/neighborhoods/neighborhood.hh

## 5.68 NeighborhoodIterator Class Reference

[Neighborhood](#) iterator.

```
#include <hnco/neighborhoods/neighborhood-iterator.hh>
```

Inheritance diagram for NeighborhoodIterator:



## Public Member Functions

- [NeighborhoodIterator](#) (int n)  
*Constructor.*
- virtual void [set\\_origin](#) (const [bit\\_vector\\_t](#) &x)  
*Set origin.*

## Additional Inherited Members

### 5.68.1 Detailed Description

[Neighborhood](#) iterator.

Definition at line 35 of file neighborhood-iterator.hh.

### 5.68.2 Constructor & Destructor Documentation

#### 5.68.2.1 NeighborhoodIterator()

```
NeighborhoodIterator (
    int n ) [inline]
```

Constructor.

#### Parameters

<i>n</i>	Size of bit vectors
----------	---------------------

Definition at line 44 of file neighborhood-iterator.hh.

The documentation for this class was generated from the following files:

- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

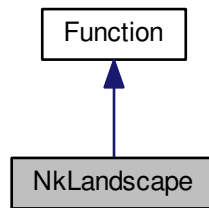
## 5.69 NkLandscape Class Reference

NK landscape.

```
#include <hnco/functions/nk-landscape.hh>
```



Inheritance diagram for NkLandscape:



### Public Member Functions

- `NkLandscape ()`  
*Default constructor.*
- `size_t get_bv_size ()`  
*Get bit vector size.*
- `void random (int n, int k, double stddev)`  
*Random instance.*
- `double eval (const bit_vector_t &)`  
*Evaluate a bit vector.*

### Private Member Functions

- `template<class Archive > void serialize (Archive &ar, const unsigned int version)`  
*Serialize.*

### Private Attributes

- `std::vector< std::vector< int > > _neighbors`  
*Bit neighbors.*
- `std::vector< std::vector< double > > _partial_functions`  
*Partial functions.*

### Friends

- class `boost::serialization::access`

#### 5.69.1 Detailed Description

NK landscape.

Reference:

S. A. Kauffman. 1993. The origins of order: self-organisation and selection in evolution. Oxford University Press.

Definition at line 47 of file `nk-landscape.hh`.

## 5.69.2 Member Function Documentation

### 5.69.2.1 random()

```
void random (
    int n,
    int k,
    double stddev )
```

Random instance.

#### Parameters

<i>n</i>	Size of bit vector
<i>k</i>	Number of neighbors of each bit
<i>stddev</i>	Standard deviation of the values of the partial functions

Definition at line 32 of file nk-landscape.cc.

The documentation for this class was generated from the following files:

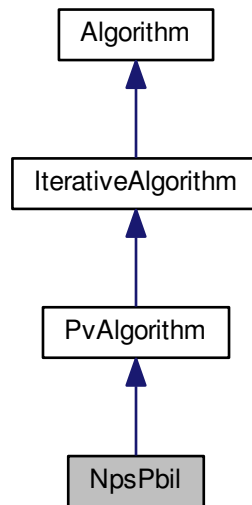
- lib/hnco/functions/nk-landscape.hh
- lib/hnco/functions/nk-landscape.cc

## 5.70 NpsPbil Class Reference

Population-based incremental learning with negative and positive selection.

```
#include <hnco/algorithms/pv/nps-pbil.hh>
```

Inheritance diagram for NpsPbil:



### Public Member Functions

- [NpsPbil](#) (int n, int population\_size)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_selection\\_size](#) (int x)  
*Set the selection size.*
- void [set\\_learning\\_rate](#) (double x)  
*Set the learning rate.*

### Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*

### Protected Attributes

- [Population \\_population](#)  
*Population.*
- [pv\\_t \\_mean\\_best](#)  
*Mean of best individuals.*
- [pv\\_t \\_mean\\_worst](#)

*Mean of worst individuals.*

### Parameters

- `int _selection_size = 1`  
*Selection size.*
- `double _learning_rate = 1e-3`  
*Learning rate.*

### Additional Inherited Members

#### 5.70.1 Detailed Description

Population-based incremental learning with negative and positive selection.

Reference:

Arnaud Berny. 2001. Extending selection learning toward fixed-length d-ary strings. In Artificial Evolution (Lecture Notes in Computer Science), P. Collet and others (Eds.). Springer, Le Creusot.

Definition at line 41 of file nps-pbil.hh.

The documentation for this class was generated from the following files:

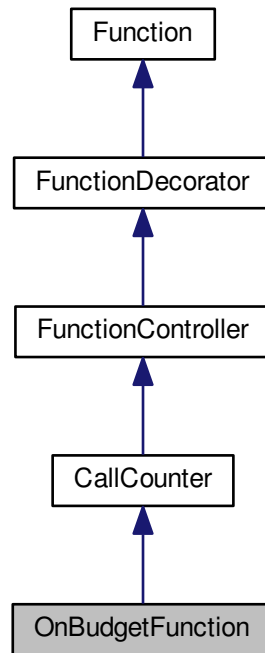
- `lib/hnco/algorithms/pv/nps-pbil.hh`
- `lib/hnco/algorithms/pv/nps-pbil.cc`

## 5.71 OnBudgetFunction Class Reference

[CallCounter](#) with a limited number of evaluations.

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for OnBudgetFunction:



## Public Member Functions

- [OnBudgetFunction](#) ([Function](#) \*function, int budget)

*Constructor.*

## Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped↔  
\_bits)  
*Incremental evaluation.*
- void [update](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Update after a safe evaluation.*

## Private Attributes

- int [\\_budget](#)  
*Budget.*

## Additional Inherited Members

### 5.71.1 Detailed Description

[CallCounter](#) with a limited number of evaluations.

Definition at line 323 of file function-controller.hh.

### 5.71.2 Member Function Documentation

#### 5.71.2.1 eval()

```
double eval (
    const bit\_vector\_t & x ) [virtual]
```

Evaluate a bit vector.

#### Exceptions

<i>LastEvaluation</i>	
-----------------------	--

Reimplemented from [CallCounter](#).

Definition at line 121 of file function-controller.cc.

#### 5.71.2.2 incremental\_eval()

```
double incremental_eval (
    const bit\_vector\_t & x,
    double value,
    const hnco::sparse\_bit\_vector\_t & flipped_bits ) [virtual]
```

Incremental evaluation.

#### Exceptions

<i>LastEvaluation</i>	
-----------------------	--

Reimplemented from [CallCounter](#).

Definition at line 132 of file function-controller.cc.

## 5.71.2.3 update()

```
void update (
    const bit\_vector\_t & x,
    double value ) [virtual]
```

Update after a safe evaluation.

## Exceptions

<i>LastEvaluation</i>	
-----------------------	--

Reimplemented from [CallCounter](#).

Definition at line 143 of file function-controller.cc.

The documentation for this class was generated from the following files:

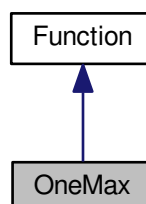
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.72 OneMax Class Reference

[OneMax](#).

```
#include <hnco/functions/theory.hh>
```

Inheritance diagram for OneMax:



## Public Member Functions

- [OneMax](#) (int bv\_size)

*Constructor.*

### Information about the function

- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- bool [provides\\_incremental\\_evaluation](#) ()  
*Check whether the function provides incremental evaluation.*

### Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double v, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped\_bits)  
*Incremental evaluation.*

## Private Attributes

- size\_t [\\_bv\\_size](#)  
*Bit vector size.*

### 5.72.1 Detailed Description

[OneMax](#).

References:

Heinz Mühlenbein, "How genetic algorithms really work: I. mutation and hillclimbing", in Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, 1992

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 40 of file theory.hh.

### 5.72.2 Member Function Documentation



### 5.72.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Returns

`_bv_size`

Reimplemented from [Function](#).

Definition at line 61 of file theory.hh.

### 5.72.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

`true`

Reimplemented from [Function](#).

Definition at line 65 of file theory.hh.

### 5.72.2.3 `provides_incremental_evaluation()`

```
bool provides_incremental_evaluation ( ) [inline], [virtual]
```

Check whether the function provides incremental evaluation.

#### Returns

`true`

Reimplemented from [Function](#).

Definition at line 70 of file theory.hh.

The documentation for this class was generated from the following files:

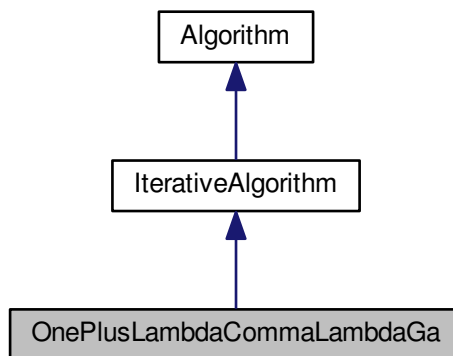
- `lib/hnco/functions/theory.hh`
- `lib/hnco/functions/theory.cc`

## 5.73 OnePlusLambdaCommaLambdaGa Class Reference

(1+(lambda, lambda)) genetic algorithm.

```
#include <hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh>
```

Inheritance diagram for OnePlusLambdaCommaLambdaGa:



### Public Member Functions

- [OnePlusLambdaCommaLambdaGa](#) (int n, int lambda)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_mutation\\_probability](#) (double x)  
*Set the mutation probability.*
- void [set\\_crossover\\_bias](#) (double x)  
*Set the crossover bias.*

### Private Member Functions

- void [iterate](#) ()  
*Single iteration.*

## Private Attributes

- [Population \\_offsprings](#)  
*Offsprings.*
- `std::binomial_distribution< int > _radius_dist`  
*Radius distribution.*
- [neighborhood::HammingSphere \\_mutation](#)  
*Mutation operator.*
- [bit\\_vector\\_t \\_parent](#)  
*Parent.*
- [BiasedCrossover \\_crossover](#)  
*Biased crossover.*

## Parameters

- `double _mutation_probability`  
*Mutation probability.*
- `double _crossover_bias`  
*Crossover bias.*

## Additional Inherited Members

### 5.73.1 Detailed Description

(1+(lambda, lambda)) genetic algorithm.

Reference:

Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. Theoretical Computer Science 567 (2015), 87–104.

Definition at line 49 of file one-plus-lambda-comma-lambda-ga.hh.

### 5.73.2 Constructor & Destructor Documentation

#### 5.73.2.1 OnePlusLambdaCommaLambdaGa()

```
OnePlusLambdaCommaLambdaGa (
    int n,
    int lambda ) [inline]
```

Constructor.

By default, `_mutation_probability` is set to `lambda / n` and `_crossover_bias` to `1 / lambda`.

## Parameters

<i>n</i>	Size of bit vectors
<i>lambda</i>	Offspring population size

Definition at line 92 of file one-plus-lambda-comma-lambda-ga.hh.

The documentation for this class was generated from the following files:

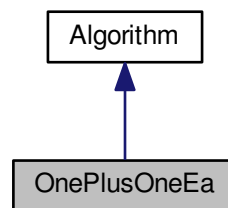
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc

## 5.74 OnePlusOneEa Class Reference

(1+1) EA.

```
#include <hnco/algorithms/ea/one-plus-one-ea.hh>
```

Inheritance diagram for OnePlusOneEa:



### Public Member Functions

- [OnePlusOneEa](#) (int n)  
*Constructor.*
- void [set\\_function](#) (function::Function \*function)  
*Set function.*
- void [init](#) ()  
*Initialization.*
- void [maximize](#) ()  
*Maximize.*
- const [point\\_value\\_t](#) & [get\\_solution](#) ()  
*Solution.*

### Setters

- void [set\\_num\\_iterations](#) (int x)  
*Set the number of iterations.*
- void [set\\_mutation\\_probability](#) (double x)  
*Set the mutation probability.*
- void [set\\_allow\\_stay](#) (bool x)  
*Set the flag\_allow\_stay.*
- void [set\\_incremental\\_evaluation](#) (bool x)  
*Set incremental evaluation.*

## Private Attributes

- [neighborhood::BernoulliProcess \\_neighborhood](#)  
*Neighborhood.*
- [RandomLocalSearch \\_rls](#)  
*Random local search.*

## Parameters

- `int _num_iterations = 0`  
*Number of iterations.*
- `double _mutation_probability`  
*Mutation probability.*
- `bool _allow_stay = false`  
*Allow stay.*
- `bool _incremental_evaluation = false`  
*Incremental evaluation.*

## Additional Inherited Members

### 5.74.1 Detailed Description

(1+1) EA.

(1+1) EA is implemented as a [RandomLocalSearch](#) with a [BernoulliProcess](#) neighborhood and infinite patience. Thus it does derive from [IterativeAlgorithm](#). It should be noted that member [Algorithm::\\_solution](#) is not used by [OnePlusOneEa](#).

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 45 of file one-plus-one-ea.hh.

### 5.74.2 Constructor & Destructor Documentation

#### 5.74.2.1 OnePlusOneEa()

```
OnePlusOneEa (
    int n ) [inline]
```

Constructor.

Parameters

<i>n</i>	Size of bit vectors
----------	---------------------

`_mutation_probability` is initialized to  $1 / n$ .

Definition at line 80 of file `one-plus-one-ea.hh`.

### 5.74.3 Member Function Documentation

#### 5.74.3.1 `set_allow_stay()`

```
void set_allow_stay (
    bool x ) [inline]
```

Set the flag `_allow_stay`.

In case no mutation occurs allow the current bit vector to stay unchanged.

Definition at line 127 of file `one-plus-one-ea.hh`.

#### 5.74.3.2 `set_num_iterations()`

```
void set_num_iterations (
    int x ) [inline]
```

Set the number of iterations.

##### Parameters

<code>x</code>	Number of iterations
----------------	----------------------

`x <= 0` means indefinite

Definition at line 117 of file `one-plus-one-ea.hh`.

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

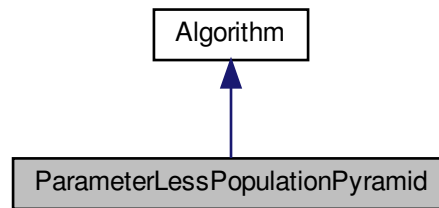
- `lib/hnco/algorithms/ea/one-plus-one-ea.hh`

## 5.75 ParameterLessPopulationPyramid Class Reference

Parameter-less Population Pyramid.

```
#include <hnco/algorithms/eda/p3.hh>
```

Inheritance diagram for ParameterLessPopulationPyramid:



### Public Member Functions

- [ParameterLessPopulationPyramid](#) (int n)  
*Constructor.*
- void [maximize](#) ()  
*Maximize.*

### Additional Inherited Members

#### 5.75.1 Detailed Description

Parameter-less Population Pyramid.

Implementation of the Parameter-less Population Pyramid (P3 for short).

Author: Brian W. Goldman

Reference:

"Fast and Efficient Black Box Optimization using the Parameter-less Population Pyramid" by B. W. Goldman and W. F. Punch

Integrated into HNCO by Arnaud Berny

Definition at line 44 of file p3.hh.

The documentation for this class was generated from the following files:

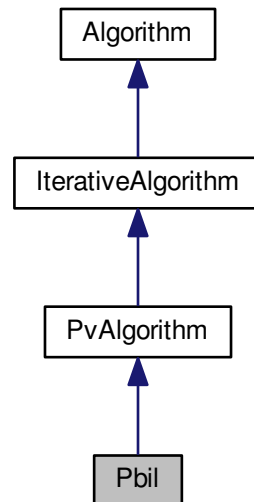
- lib/hnco/algorithms/eda/p3.hh
- lib/hnco/algorithms/eda/p3.cc

## 5.76 Pbil Class Reference

Population-based incremental learning.

```
#include <hnco/algorithms/pv/pbil.hh>
```

Inheritance diagram for Pbil:



### Public Member Functions

- [Pbil](#) (int n, int population\_size)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_selection\\_size](#) (int x)  
*Set the selection size.*
- void [set\\_learning\\_rate](#) (double x)  
*Set the learning rate.*

### Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*



## Protected Attributes

- [Population \\_population](#)  
*Population.*
- [pv\\_t \\_mean](#)  
*Mean of selected bit vectors.*

## Parameters

- `int _selection_size = 1`  
*Selection size.*
- `double _learning_rate = 1e-3`  
*Learning rate.*

## Additional Inherited Members

### 5.76.1 Detailed Description

Population-based incremental learning.

Reference:

S. Baluja and R. Caruana. 1995. Removing the genetics from the standard genetic algorithm. In Proceedings of the 12th Annual Conference on Machine Learning. 38–46.

Definition at line 40 of file pbil.hh.

The documentation for this class was generated from the following files:

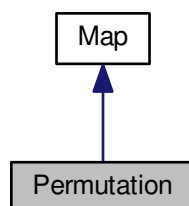
- `lib/hnco/algorithms/pv/pbil.hh`
- `lib/hnco/algorithms/pv/pbil.cc`

## 5.77 Permutation Class Reference

[Permutation.](#)

```
#include <hnco/map.hh>
```

Inheritance diagram for Permutation:



## Public Member Functions

- void [random](#) (int n)  
*Random instance.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- size\_t [get\\_input\\_size](#) ()  
*Get input size.*
- size\_t [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

## Private Member Functions

- template<class Archive >  
void [save](#) (Archive &ar, const unsigned int version) const  
*Save.*
- template<class Archive >  
void [load](#) (Archive &ar, const unsigned int version)  
*Load.*

## Private Attributes

- [permutation\\_t \\_permutation](#)  
*Permutation.*

## Friends

- class **boost::serialization::access**

### 5.77.1 Detailed Description

#### [Permutation.](#)

A permutation is a linear map  $f$  from  $Z_2^n$  to itself defined by  $f(x) = y$ , where  $y_i = x_{\sigma_i}$  and  $\sigma$  is a permutation of 0, 1, ..., n - 1.

Definition at line 132 of file map.hh.

### 5.77.2 Member Function Documentation

### 5.77.2.1 is\_surjective()

```
bool is_surjective ( ) [inline], [virtual]
```

Check for surjective map.

#### Returns

true

Reimplemented from [Map](#).

Definition at line 183 of file map.hh.

The documentation for this class was generated from the following files:

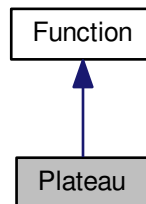
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.78 Plateau Class Reference

[Plateau](#).

```
#include <hnco/functions/theory.hh>
```

Inheritance diagram for Plateau:



### Public Member Functions

- [Plateau](#) (int bv\_size)  
*Constructor.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- bool [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- double [get\\_maximum](#) ()  
*Get the global maximum.*

## Private Attributes

- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*

### 5.78.1 Detailed Description

[Plateau](#).

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 243 of file theory.hh.

### 5.78.2 Member Function Documentation

#### 5.78.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

Returns

`_bv_size + 2`

Reimplemented from [Function](#).

Definition at line 267 of file theory.hh.

#### 5.78.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

Returns

`true`

Reimplemented from [Function](#).

Definition at line 263 of file theory.hh.

The documentation for this class was generated from the following files:

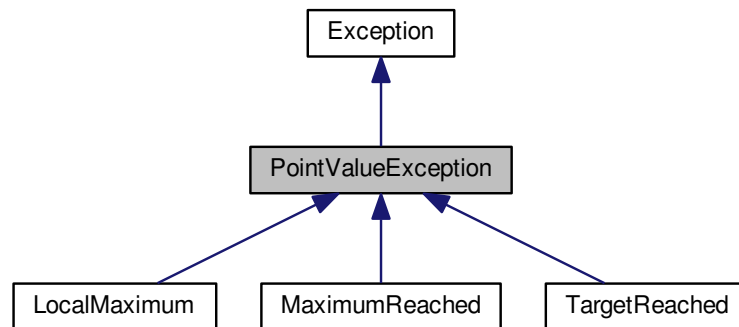
- `lib/hnco/functions/theory.hh`
- `lib/hnco/functions/theory.cc`

## 5.79 PointValueException Class Reference

Point-value exception.

```
#include <hnco/exception.hh>
```

Inheritance diagram for PointValueException:



### Public Member Functions

- [PointValueException](#) (const [point\\_value\\_t](#) &pv)  
*Constructor.*
- const [point\\_value\\_t](#) & [get\\_point\\_value](#) () const  
*Get point-value.*

### Protected Attributes

- [point\\_value\\_t \\_pv](#)  
*Point-value.*

#### 5.79.1 Detailed Description

Point-value exception.

Definition at line 38 of file exception.hh.

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

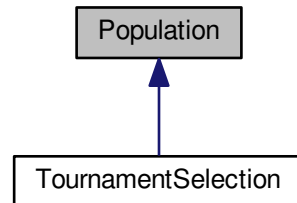
- lib/hnco/exception.hh

## 5.80 Population Class Reference

Population.

```
#include <hnco/algorithms/population.hh>
```

Inheritance diagram for Population:



### Public Types

- `typedef std::pair< size_t, double > index\_value\_t`  
*Index-value type.*

### Public Member Functions

- [Population](#) (int population\_size, int n)  
*Constructor.*
- `std::size_t size () const`  
*Size.*
- `void random ()`  
*Initialize the population with random bit vectors.*

#### Get bit vectors for non const populations

- `bit\_vector\_t & get\_bv (int i)`  
*Get a bit vector.*
- `bit\_vector\_t & get\_best\_bv ()`  
*Get best bit vector.*
- `bit\_vector\_t & get\_best\_bv (int i)`  
*Get best bit vector.*
- `bit\_vector\_t & get\_worst\_bv (int i)`  
*Get worst bit vector.*

#### Get bit vectors for const populations

- `const bit\_vector\_t & get\_bv (int i) const`  
*Get a bit vector.*
- `const bit\_vector\_t & get\_best\_bv () const`

- *Get best bit vector.*  
const [bit\\_vector\\_t](#) & [get\\_best\\_bv](#) (int i) const
- *Get best bit vector.*  
const [bit\\_vector\\_t](#) & [get\\_worst\\_bv](#) (int i) const
- *Get worst bit vector.*

### Get sorted values

- double [get\\_best\\_value](#) (int i) const  
*Get best value.*
- double [get\\_best\\_value](#) () const  
*Get best value.*

### Evaluation and sorting

- void [eval](#) (function::Function \*function)  
*Evaluate the population.*
- void [eval](#) (const std::vector< function::Function \*> &functions)  
*Parallel evaluation of the population.*
- void [sort](#) ()  
*Sort the lookup table.*
- void [partial\\_sort](#) (int selection\_size)  
*Partially sort the lookup table.*
- void [shuffle](#) ()  
*Shuffle the lookup table.*

### Selection

- void [plus\\_selection](#) (const Population &offsprings)  
*Plus selection.*
- void [plus\\_selection](#) (Population &offsprings)  
*Plus selection.*
- void [comma\\_selection](#) (const Population &offsprings)  
*Comma selection.*
- void [comma\\_selection](#) (Population &offsprings)  
*Comma selection.*

### Protected Attributes

- std::vector< [bit\\_vector\\_t](#) > [\\_bvs](#)  
*Bit vectors.*
- std::vector< [index\\_value\\_t](#) > [\\_lookup](#)  
*Lookup table.*
- std::function< bool(const [index\\_value\\_t](#) &, const [index\\_value\\_t](#) &)> [\\_compare\\_index\\_value](#)  
*Binary operator for comparing index-value pairs.*

## 5.80.1 Detailed Description

[Population](#).

Definition at line 36 of file population.hh.

## 5.80.2 Member Function Documentation

### 5.80.2.1 comma\_selection() [1/2]

```
void comma_selection (
    const Population & offsprings )
```

Comma selection.

Implemented with a copy.

#### Precondition

Offspring population must be partially sorted.

#### Warning

The function does not break ties randomly (workaround: shuffle offsprings).

Definition at line 112 of file population.cc.

### 5.80.2.2 comma\_selection() [2/2]

```
void comma_selection (
    Population & offsprings )
```

Comma selection.

Implemented with a swap. Should be faster than comma\_selection with a copy.

#### Precondition

Offspring population must be partially sorted.

#### Warning

The function does not break ties randomly (workaround: shuffle offsprings).  
Modifies its argument.

Definition at line 126 of file population.cc.



#### 5.80.2.3 `get_best_bv()` [1/4]

```
bit_vector_t& get_best_bv ( ) [inline]
```

Get best bit vector.

##### Precondition

The population must be sorted.

Definition at line 85 of file population.hh.

#### 5.80.2.4 `get_best_bv()` [2/4]

```
bit_vector_t& get_best_bv (
    int i ) [inline]
```

Get best bit vector.

##### Parameters

<i>i</i>	Index in the sorted population
----------	--------------------------------

##### Precondition

The population must be sorted.

Definition at line 93 of file population.hh.

#### 5.80.2.5 `get_best_bv()` [3/4]

```
const bit_vector_t& get_best_bv ( ) const [inline]
```

Get best bit vector.

##### Precondition

The population must be sorted.

Definition at line 117 of file population.hh.

#### 5.80.2.6 `get_best_bv()` [4/4]

```
const bit_vector_t& get_best_bv (
    int i ) const [inline]
```

Get best bit vector.

**Parameters**

<i>i</i>	Index in the sorted population
----------	--------------------------------

**Precondition**

The population must be sorted.

Definition at line 125 of file population.hh.

**5.80.2.7 get\_best\_value()** [1/2]

```
double get_best_value (  
    int i ) const [inline]
```

Get best value.

**Parameters**

<i>i</i>	Index in the sorted population
----------	--------------------------------

**Precondition**

The population must be sorted.

Definition at line 148 of file population.hh.

**5.80.2.8 get\_best\_value()** [2/2]

```
double get_best_value ( ) const [inline]
```

Get best value.

**Precondition**

The population must be sorted.

Definition at line 154 of file population.hh.

**5.80.2.9 get\_worst\_bv()** [1/2]

```
bit_vector_t& get_worst_bv (  
    int i ) [inline]
```

Get worst bit vector.

**Parameters**

<i>i</i>	Index in the sorted population
----------	--------------------------------

**Precondition**

The population must be sorted.

Definition at line 101 of file population.hh.

**5.80.2.10 get\_worst\_bv()** [2/2]

```
const bit_vector_t& get_worst_bv (  
    int i ) const [inline]
```

Get worst bit vector.

**Parameters**

<i>i</i>	Index in the sorted population
----------	--------------------------------

**Precondition**

The population must be sorted.

Definition at line 133 of file population.hh.

**5.80.2.11 plus\_selection()** [1/2]

```
void plus_selection (  
    const Population & offsprings )
```

Plus selection.

Implemented with a copy.

**Precondition**

Both populations must be completely sorted.

**Warning**

The function does not break ties randomly (workaround: shuffle parents and offsprings).

Definition at line 74 of file population.cc.

#### 5.80.2.12 plus\_selection() [2/2]

```
void plus_selection (
    Population & offsprings )
```

Plus selection.

Implemented with a swap. Should be faster than plus\_selection with a copy.

##### Precondition

Both populations must be completely sorted.

##### Warning

The function does not break ties randomly (workaround: shuffle parents and offsprings).  
Modifies its argument.

Definition at line 93 of file population.cc.

### 5.80.3 Member Data Documentation

#### 5.80.3.1 \_compare\_index\_value

```
std::function<bool(const index_value_t&, const index_value_t&)> _compare_index_value [protected]
```

##### Initial value:

```
=
    [](const index_value_t& a, const index_value_t& b) { return a.second > b.
    second; }
```

Binary operator for comparing index-value pairs.

Definition at line 57 of file population.hh.

#### 5.80.3.2 \_lookup

```
std::vector<index_value_t> _lookup [protected]
```

Lookup table.

Let p be of type std::pair<size\_t, double>. Then p.first is the bv index in the unsorted population whereas p.second is the bv value.

Definition at line 54 of file population.hh.

The documentation for this class was generated from the following files:

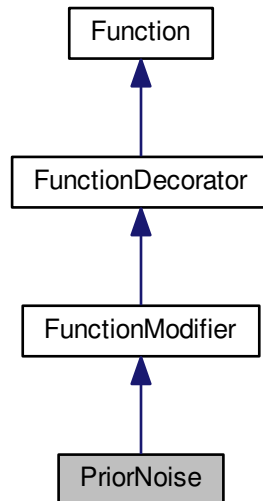
- lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

## 5.81 PriorNoise Class Reference

Prior noise.

```
#include <hnco/functions/decorators/prior-noise.hh>
```

Inheritance diagram for PriorNoise:



### Public Member Functions

- [PriorNoise](#) ([Function](#) \*fn, [neighborhood::Neighborhood](#) \*nh)  
*Constructor.*

### Information about the function

- [size\\_t](#) [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double](#) [get\\_maximum](#) ()  
*Get the global maximum.*
- [bool](#) [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [bool](#) [provides\\_incremental\\_evaluation](#) ()  
*Check whether the function provides incremental evaluation.*

### Evaluation

- [double](#) [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Private Attributes

- [neighborhood::Neighborhood](#) \* [\\_neighborhood](#)  
*Neighborhood.*
- [bit\\_vector\\_t](#) [\\_noisy\\_bv](#)  
*Noisy bit vector.*

## Additional Inherited Members

### 5.81.1 Detailed Description

Prior noise.

Definition at line 35 of file prior-noise.hh.

### 5.81.2 Member Function Documentation

#### 5.81.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

Delegation is questionable here.

Reimplemented from [Function](#).

Definition at line 67 of file prior-noise.hh.

#### 5.81.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

Delegation is questionable here.

Reimplemented from [Function](#).

Definition at line 73 of file prior-noise.hh.

### 5.81.2.3 provides\_incremental\_evaluation()

```
bool provides_incremental_evaluation ( ) [inline], [virtual]
```

Check whether the function provides incremental evaluation.

#### Returns

false

Reimplemented from [Function](#).

Definition at line 77 of file prior-noise.hh.

The documentation for this class was generated from the following files:

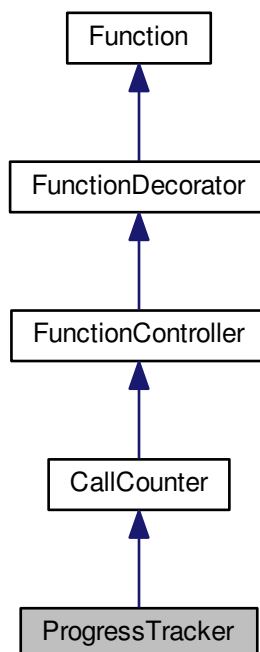
- lib/hnco/functions/decorators/prior-noise.hh
- lib/hnco/functions/decorators/prior-noise.cc

## 5.82 ProgressTracker Class Reference

[ProgressTracker](#).

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for ProgressTracker:



## Classes

- struct [Event](#)  
*Event.*

## Public Member Functions

- [ProgressTracker](#) ([Function](#) \*function)  
*Constructor.*

## Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped←\_bits)  
*Incremental evaluation.*
- void [update](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Update after a safe evaluation.*

## Get information

- const [Event](#) & [get\\_last\\_improvement](#) ()  
*Get the last improvement.*
- double [get\\_evaluation\\_time](#) ()  
*Get evaluation time.*

## Setters

- void [set\\_log\\_improvement](#) (bool x)  
*Log improvement.*
- void [set\\_stream](#) (std::ostream \*x)  
*Output stream.*

## Protected Member Functions

- void [update\\_last\\_improvement](#) (double value)  
*Update last improvement.*

## Protected Attributes

- [Event\\_last\\_improvement](#)  
*Last improvement.*
- [StopWatch\\_stop\\_watch](#)  
*Stop watch.*

## Parameters

- bool [\\_log\\_improvement](#) = false  
*Log improvement.*
- std::ostream \* [\\_stream](#) = &std::cout  
*Output stream.*



### 5.82.1 Detailed Description

[ProgressTracker](#).

A [ProgressTracker](#) is a [CallCounter](#) which keeps track the last improvement, that is its value and the number of evaluations needed to reach it.

Definition at line 213 of file function-controller.hh.

### 5.82.2 Member Function Documentation

#### 5.82.2.1 eval()

```
double eval (
    const bit\_vector\_t & x ) [virtual]
```

Evaluate a bit vector.

#### Exceptions

<i>MaximumReached</i>	
<i>TargetReached</i>	

Reimplemented from [CallCounter](#).

Definition at line 153 of file function-controller.cc.

#### 5.82.2.2 get\_last\_improvement()

```
const Event& get_last_improvement ( ) [inline]
```

Get the last improvement.

#### Warning

If `_last_improvement.num_evaluations` is zero then `_function` has never been called. The [Event](#) returned by `get_last_improvement` has therefore no meaning.

Definition at line 296 of file function-controller.hh.

#### 5.82.2.3 incremental\_eval()

```
double incremental_eval (
    const bit\_vector\_t & x,
    double value,
    const hnco::sparse\_bit\_vector\_t & flipped_bits ) [virtual]
```

Incremental evaluation.

## Exceptions

<i>MaximumReached</i>	
<i>TargetReached</i>	

Reimplemented from [CallCounter](#).

Definition at line 176 of file function-controller.cc.

## 5.82.2.4 update()

```
void update (
    const bit\_vector\_t & x,
    double value ) [virtual]
```

Update after a safe evaluation.

## Exceptions

<i>MaximumReached</i>	
<i>TargetReached</i>	

Reimplemented from [CallCounter](#).

Definition at line 195 of file function-controller.cc.

The documentation for this class was generated from the following files:

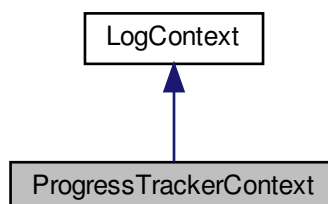
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.83 ProgressTrackerContext Class Reference

Log context for ProgressTracker.

```
#include <hnco/algorithms/log-context.hh>
```

Inheritance diagram for ProgressTrackerContext:



## Public Member Functions

- [ProgressTrackerContext](#) ([hnco::function::ProgressTracker](#) \*pt)  
*Constructor.*
- `std::string` [get\\_context](#) ()  
*Get context.*

## Private Attributes

- [hnco::function::ProgressTracker](#) \* [\\_pt](#)  
*Progress tracker.*

### 5.83.1 Detailed Description

Log context for ProgressTracker.

Definition at line 48 of file log-context.hh.

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

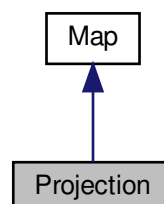
- lib/hnco/algorithms/log-context.hh

## 5.84 Projection Class Reference

[Projection](#).

```
#include <hnco/map.hh>
```

Inheritance diagram for Projection:



## Public Member Functions

- [Projection](#) (const std::vector< std::size\_t > &bit\_positions, std::size\_t input\_size)  
*Constructor.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- size\_t [get\\_input\\_size](#) ()  
*Get input size.*
- size\_t [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

## Private Attributes

- std::vector< std::size\_t > [\\_bit\\_positions](#)  
*Bit positions.*
- std::size\_t [\\_input\\_size](#)  
*Input size.*

### 5.84.1 Detailed Description

#### [Projection.](#)

The projection  $y$  of a bit vector  $x$  is  $x$  where we have dropped a given set of components.

Let  $I = \{i_1, i_2, \dots, i_m\}$  be a subset of  $\{1, 2, \dots, n\}$ .

A projection  $f$  from  $Z_2^n$  to  $Z_2^m$ , where  $n \geq m$ , is defined by  $f(x) = y$ , where, for all  $j \in \{1, 2, \dots, m\}$ ,  $y_j = x_{i_j}$ .

If  $f$  is a projection and  $g$  is an injection with the same bit positions then their composition  $f \circ g$  is the identity.

Definition at line 452 of file map.hh.

### 5.84.2 Constructor & Destructor Documentation

#### 5.84.2.1 [Projection\(\)](#)

```
Projection (
    const std::vector< std::size_t > & bit_positions,
    std::size_t input_size )
```

Constructor.

The output size of the map is given by the size of `bit_positions`.

## Parameters

<i>bit_positions</i>	Bit positions in the input from where output bits are copied
<i>input_size</i>	Input size

## Precondition

`input_size >= bit_positions.size()`

Definition at line 164 of file map.cc.

### 5.84.3 Member Function Documentation

#### 5.84.3.1 is\_surjective()

```
bool is_surjective ( ) [inline], [virtual]
```

Check for surjective map.

## Returns

`true`

Reimplemented from [Map](#).

Definition at line 490 of file map.hh.

The documentation for this class was generated from the following files:

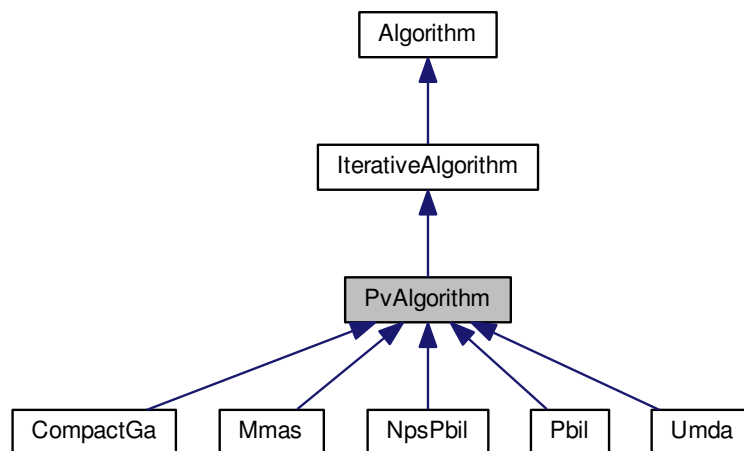
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.85 PvAlgorithm Class Reference

Probability vector algorithm.

```
#include <hnco/algorithms/pv/pv-algorithm.hh>
```

Inheritance diagram for PvAlgorithm:



## Public Types

- enum { [LOG\\_PV](#), [LOG\\_ENTROPY](#), [LAST\\_LOG](#) }
- typedef std::bitset< LAST\_LOG > [log\\_flags\\_t](#)

## Public Member Functions

- [PvAlgorithm](#) (int n)  
*Constructor.*
- void [set\\_log\\_flags](#) (const [log\\_flags\\_t](#) &lf)  
*Set log flags.*

## Setters

- void [set\\_log\\_num\\_components](#) (int x)  
*Set the number of probability vector components to log.*

## Protected Member Functions

- void [log](#) ()  
*Log.*

## Protected Attributes

- `pv_t _pv`  
*Probability vector.*
- `double _lower_bound`  
*Lower bound of probability.*
- `double _upper_bound`  
*Upper bound of probability.*
- `log_flags_t _log_flags`  
*Log flags.*

## Parameters

- `int _log_num_components = 5`  
*Number of probability vector components to log.*

### 5.85.1 Detailed Description

Probability vector algorithm.

Definition at line 34 of file `pv-algorithm.hh`.

### 5.85.2 Member Enumeration Documentation

#### 5.85.2.1 anonymous enum

`anonymous enum`

#### Enumerator

<code>LOG_PV</code>	Log probability vector.
<code>LOG_ENTROPY</code>	Log entropy.

Definition at line 39 of file `pv-algorithm.hh`.

The documentation for this class was generated from the following files:

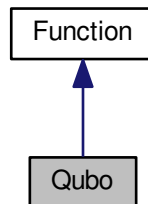
- `lib/hnco/algorithms/pv/pv-algorithm.hh`
- `lib/hnco/algorithms/pv/pv-algorithm.cc`

## 5.86 Qubo Class Reference

Quadratic unconstrained binary optimization.

```
#include <hnco/functions/qubo.hh>
```

Inheritance diagram for Qubo:



## Public Member Functions

- [Qubo](#) ()  
*Constructor.*
- void [load](#) (std::istream &stream)  
*Load an instance.*
- size\_t [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Private Attributes

- std::vector< std::vector< double > > [\\_q](#)  
*Matrix.*

### 5.86.1 Detailed Description

Quadratic unconstrained binary optimization.

Its expression is of the form  $f(x) = \sum_i Q_{ii}x_i + \sum_{i < j} Q_{ij}x_ix_{ij} = x^T Qx$ , where Q is an n x n upper-triangular matrix.

[Qubo](#) is the problem addressed by qbsolv. Here is its description as given on github:

Qbsolv, a decomposing solver, finds a minimum value of a large quadratic unconstrained binary optimization (QUBO) problem by splitting it into pieces solved either via a D-Wave system or a classical tabu solver.

There are some differences between [WalshExpansion2](#) and [Qubo](#):

- [WalshExpansion2](#) maps 0/1 variables into -1/1 variables whereas [Qubo](#) directly deals with binary variables.



- Hence, there is a separate linear part in [WalshExpansion2](#) whereas the linear part in [Qubo](#) stems from the diagonal elements of the given matrix.

qbsolv aims at minimizing quadratic functions whereas hnco algorithms aim at maximizing them. Hence [Qubo::load](#) negates all elements so that maximizing the resulting function is equivalent to minimizing the original [Qubo](#).

References:

Michael Booth, Steven P. Reinhardt, and Aidan Roy. 2017. Partitioning Optimization Problems for Hybrid Classical/Quantum Execution. Technical Report. D-Wave.

<https://github.com/dwavesystems/qbsolv>

<http://people.brunel.ac.uk/~mastjjb/jeb/orlib/bqpinfo.html>

Definition at line 74 of file qubo.hh.

## 5.86.2 Member Function Documentation

### 5.86.2.1 load()

```
void load (
    std::istream & stream )
```

Load an instance.

Exceptions

Error	
-------	--

Definition at line 35 of file qubo.cc.

## 5.86.3 Member Data Documentation

### 5.86.3.1 \_q

```
std::vector<std::vector<double> > _q [private]
```

Matrix.

n x n upper triangular matrix.

Definition at line 83 of file qubo.hh.

The documentation for this class was generated from the following files:

- lib/hnco/functions/qubo.hh
- lib/hnco/functions/qubo.cc

## 5.87 Random Struct Reference

Random numbers.

```
#include <hnco/random.hh>
```

### Static Public Member Functions

- static double [uniform](#) ()  
*Next uniformly distributed sample.*
- static double [normal](#) ()  
*Next normally distributed sample.*
- static bool [bernoulli](#) ()  
*Next random bit.*

### Static Public Attributes

- static std::mt19937 [engine](#)  
*Engine.*

### 5.87.1 Detailed Description

Random numbers.

Definition at line 33 of file random.hh.

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

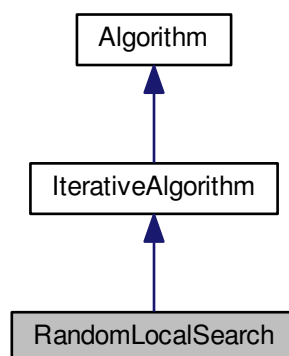
- lib/hnco/random.hh
- lib/hnco/random.cc

## 5.88 RandomLocalSearch Class Reference

Random local search.

```
#include <hnco/algorithms/ls/random-local-search.hh>
```

Inheritance diagram for RandomLocalSearch:



## Public Member Functions

- [RandomLocalSearch](#) (int n, [neighborhood::Neighborhood](#) \*neighborhood)  
*Constructor.*
- void [init](#) ()  
*Random initialization.*
- void [init](#) (const [bit\\_vector\\_t](#) &x)  
*Explicit initialization.*
- void [init](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Explicit initialization.*
- const [point\\_value\\_t](#) & [get\\_solution](#) ()  
*Solution.*

## Setters

- void [set\\_compare](#) (std::function< bool(double, double)> x)  
*Set the binary operator for comparing evaluations.*
- void [set\\_patience](#) (int x)  
*Set patience.*
- void [set\\_incremental\\_evaluation](#) (bool x)  
*Set incremental evaluation.*

## Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*
- void [iterate\\_full](#) ()  
*Single iteration with full evaluation.*
- void [iterate\\_incremental](#) ()  
*Single iteration with incremental evaluation.*

## Protected Attributes

- [neighborhood::Neighborhood](#) \* [\\_neighborhood](#)  
*Neighborhood.*
- int [\\_num\\_failures](#)  
*Number of failure.*

## Parameters

- std::function< bool(double, double)> [\\_compare](#) = std::greater\_equal<double>()  
*Binary operator for comparing evaluations.*
- int [\\_patience](#) = 50  
*Patience.*
- bool [\\_incremental\\_evaluation](#) = false  
*Incremental evaluation.*

### 5.88.1 Detailed Description

Random local search.

Definition at line 39 of file random-local-search.hh.

## 5.88.2 Member Function Documentation

### 5.88.2.1 set\_patience()

```
void set_patience (
    int x ) [inline]
```

Set patience.

Number of consecutive rejected moves before throwing a LocalMaximum exception

#### Parameters

x	Patience
---	----------

If  $x \leq 0$  then patience is considered infinite, meaning that the algorithm will never throw any LocalMaximum exception.

Definition at line 110 of file random-local-search.hh.

The documentation for this class was generated from the following files:

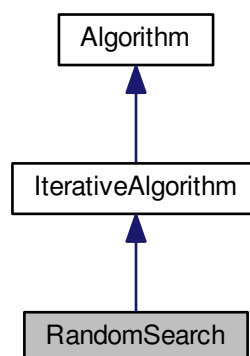
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

## 5.89 RandomSearch Class Reference

Random search.

```
#include <hnco/algorithms/random-search.hh>
```

Inheritance diagram for RandomSearch:



### Public Member Functions

- [RandomSearch](#) (int n)  
*Constructor.*

### Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*

### Private Attributes

- [bit\\_vector\\_t \\_candidate](#)  
*Candidate.*

### Additional Inherited Members

#### 5.89.1 Detailed Description

Random search.

Definition at line 30 of file random-search.hh.

The documentation for this class was generated from the following files:

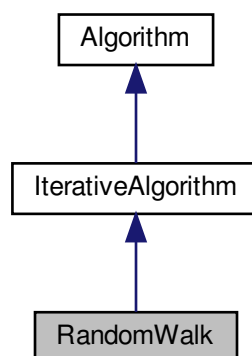
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

## 5.90 RandomWalk Class Reference

Random walk.

```
#include <hnco/algorithms/ls/random-walk.hh>
```

Inheritance diagram for RandomWalk:



## Public Member Functions

- [RandomWalk](#) (int n, [neighborhood::Neighborhood](#) \*neighborhood)  
*Constructor.*
- void [init](#) ()  
*Random initialization.*
- void [init](#) (const [bit\\_vector\\_t](#) &x)  
*Explicit initialization.*
- void [init](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Explicit initialization.*
- void [log](#) ()  
*Log.*

## Setters

- void [set\\_incremental\\_evaluation](#) (bool x)  
*Set incremental evaluation.*
- void [set\\_log\\_value](#) ()  
*Set log.*

## Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*
- void [iterate\\_full](#) ()  
*Single iteration with full evaluation.*
- void [iterate\\_incremental](#) ()  
*Single iteration with incremental evaluation.*

## Protected Attributes

- [neighborhood::Neighborhood](#) \* [\\_neighborhood](#)  
*Neighborhood.*
- double [\\_value](#)  
*Value of the last visited bit vector.*

## Parameters

- bool [\\_incremental\\_evaluation](#) = false  
*Incremental evaluation.*

### 5.90.1 Detailed Description

Random walk.

The algorithm simply performs a random walk on the graph implicitly given by the neighborhood. At each iteration, the chosen neighbor does not depend on its evaluation. However optimization takes place as in random search, that is the best visited bit vector is remembered.

Definition at line 42 of file random-walk.hh.

The documentation for this class was generated from the following files:

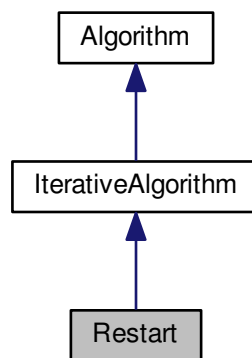
- lib/hnco/algorithms/ls/random-walk.hh
- lib/hnco/algorithms/ls/random-walk.cc

## 5.91 Restart Class Reference

[Restart](#).

```
#include <hnco/algorithms/decorators/restart.hh>
```

Inheritance diagram for Restart:



### Public Member Functions

- [Restart](#) (int n, [Algorithm](#) \*algorithm)  
*Constructor.*
- void [init](#) ()  
*Initialization.*
- void [set\\_function](#) (function::Function \*function)  
*Set function.*
- void [set\\_functions](#) (const std::vector< [function::Function](#) \*> functions)  
*Set functions.*

### Private Member Functions

- void [iterate](#) ()  
*Optimize.*

### Private Attributes

- [Algorithm](#) \* [\\_algorithm](#)  
*Algorithm.*

## Additional Inherited Members

### 5.91.1 Detailed Description

[Restart](#).

[Restart](#) an [Algorithm](#) an indefinite number of times. Should be used in conjunction with `OnBudgetFunction` or `StopOnMaximum`.

Definition at line 38 of file `restart.hh`.

The documentation for this class was generated from the following files:

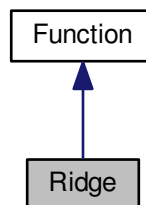
- `lib/hnco/algorithms/decorators/restart.hh`
- `lib/hnco/algorithms/decorators/restart.cc`

## 5.92 Ridge Class Reference

[Ridge](#).

```
#include <hnco/functions/theory.hh>
```

Inheritance diagram for `Ridge`:



### Public Member Functions

- [Ridge](#) (int `bv_size`)  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `double` [eval](#) (const `bit_vector_t` &)  
*Evaluate a bit vector.*
- `bool` [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- `double` [get\\_maximum](#) ()  
*Get the global maximum.*



## Private Attributes

- `size_t _bv_size`  
*Bit vector size.*

### 5.92.1 Detailed Description

[Ridge](#).

Reference:

Thomas Jansen, Analyzing Evolutionary Algorithms. Springer, 2013.

Definition at line 207 of file theory.hh.

### 5.92.2 Member Function Documentation

#### 5.92.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

Returns

`2 * _bv_size`

Reimplemented from [Function](#).

Definition at line 231 of file theory.hh.

#### 5.92.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

Returns

`true`

Reimplemented from [Function](#).

Definition at line 227 of file theory.hh.

The documentation for this class was generated from the following files:

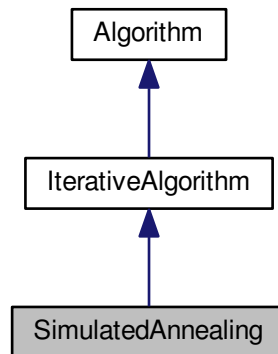
- `lib/hnco/functions/theory.hh`
- `lib/hnco/functions/theory.cc`

## 5.93 SimulatedAnnealing Class Reference

Simulated annealing.

```
#include <hnco/algorithms/ls/simulated-annealing.hh>
```

Inheritance diagram for SimulatedAnnealing:



### Public Member Functions

- [SimulatedAnnealing](#) (int n, [neighborhood::Neighborhood](#) \*neighborhood)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_num\\_transitions](#) (int x)  
*Set the number of accepted transitions before annealing.*
- void [set\\_num\\_trials](#) (int x)  
*Set the Number of trials.*
- void [set\\_initial\\_acceptance\\_probability](#) (double x)  
*Set the initial acceptance probability.*
- void [set\\_beta\\_ratio](#) (double x)  
*Set ratio for beta.*

### Private Member Functions

- void [init\\_beta](#) ()  
*Initialize beta.*
- void [iterate](#) ()  
*Single iteration.*

## Private Attributes

- `neighborhood::Neighborhood * _neighborhood`  
*Neighborhood.*
- `double _beta`  
*Inverse temperature.*
- `double _current_value`  
*Current value.*
- `int _transitions`  
*Number of accepted transitions.*

## Parameters

- `int _num_transitions = 50`  
*Number of accepted transitions before annealing.*
- `int _num_trials = 100`  
*Number of trials.*
- `double _initial_acceptance_probability = 0.6`  
*Initial acceptance probability.*
- `double _beta_ratio = 1.2`  
*Ratio for beta.*

## Additional Inherited Members

### 5.93.1 Detailed Description

Simulated annealing.

Reference:

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (May 1983), 671–680.

Definition at line 44 of file `simulated-annealing.hh`.

### 5.93.2 Member Function Documentation

#### 5.93.2.1 `init_beta()`

```
void init_beta ( ) [private]
```

Initialize beta.

Requires  $(2 * \text{\_num\_trials})$  evaluations. This should be taken into account when using `OnBudgetFunction`.

Definition at line 34 of file `simulated-annealing.cc`.

The documentation for this class was generated from the following files:

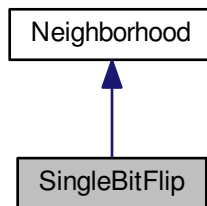
- `lib/hnco/algorithms/ls/simulated-annealing.hh`
- `lib/hnco/algorithms/ls/simulated-annealing.cc`

## 5.94 SingleBitFlip Class Reference

One bit neighborhood.

```
#include <hnco/neighborhoods/neighborhood.hh>
```

Inheritance diagram for SingleBitFlip:



### Public Member Functions

- [SingleBitFlip](#) (int n)  
*Constructor.*

### Private Member Functions

- void [sample\\_bits](#) ()  
*Sample bits.*

### Additional Inherited Members

#### 5.94.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

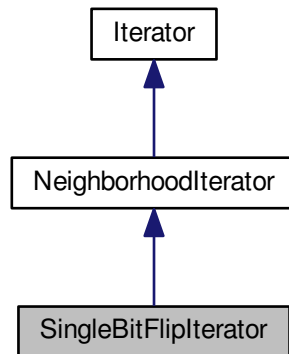
- lib/hnco/neighborhoods/neighborhood.hh

## 5.95 SingleBitFlipterator Class Reference

Single bit flip neighborhood iterator.

```
#include <hnco/neighborhoods/neighborhood-iterator.hh>
```

Inheritance diagram for SingleBitFlipterator:



### Public Member Functions

- [SingleBitFlipterator](#) (int n)  
*Constructor.*
- bool [has\\_next](#) ()  
*Has next bit vector.*
- const [bit\\_vector\\_t](#) & [next](#) ()  
*Next bit vector.*

### Private Attributes

- [size\\_t](#) [\\_index](#)  
*Index of the last flipped bit.*

### Additional Inherited Members

#### 5.95.1 Detailed Description

Single bit flip neighborhood iterator.

Definition at line 53 of file neighborhood-iterator.hh.

## 5.95.2 Constructor & Destructor Documentation

### 5.95.2.1 SingleBitFlipIterator()

```
SingleBitFlipIterator (
    int n ) [inline]
```

Constructor.

Parameters

$n$	Size of bit vectors
-----	---------------------

Definition at line 65 of file neighborhood-iterator.hh.

The documentation for this class was generated from the following files:

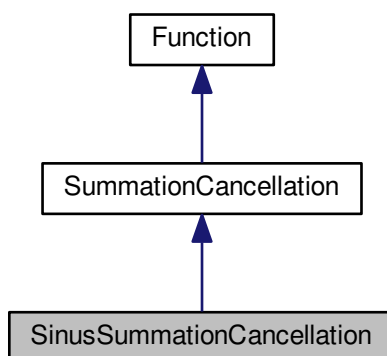
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

## 5.96 SinusSummationCancellation Class Reference

Summation cancellation with sinus.

```
#include <hnco/functions/cancellation.hh>
```

Inheritance diagram for SinusSummationCancellation:



## Public Member Functions

- [SinusSummationCancellation](#) (int n)  
*Constructor.*
- double [eval](#) (const [bit\\_vector\\_t](#) &x)  
*Evaluate a bit vector.*

## Additional Inherited Members

### 5.96.1 Detailed Description

Summation cancellation with sinus.

Reference:

M. Sebag and M. Schoenauer. 1997. A society of hill-climbers. In Proc. IEEE Int. Conf. on Evolutionary Computation. Indianapolis, 319–324.

Definition at line 103 of file `cancellation.hh`.

The documentation for this class was generated from the following files:

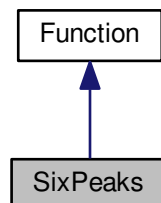
- `lib/hnco/functions/cancellation.hh`
- `lib/hnco/functions/cancellation.cc`

## 5.97 SixPeaks Class Reference

Six Peaks.

```
#include <hnco/functions/four-peaks.hh>
```

Inheritance diagram for SixPeaks:



## Public Member Functions

- [SixPeaks](#) (int bv\_size, int threshold)  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `double` [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- `bool` [has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- `double` [get\\_maximum](#) ()  
*Get the global maximum.*

## Private Attributes

- `size_t` [\\_bv\\_size](#)  
*Bit vector size.*
- `int` [\\_threshold](#)  
*Threshold.*
- `int` [\\_maximum](#)  
*Maximum.*

### 5.97.1 Detailed Description

Six Peaks.

It is defined by

$$f(x) = \max\{\text{head}(x, 0) + \text{tail}(x, 1) + \text{head}(x, 1) + \text{tail}(x, 0)\} + R(x)$$

where:

- $\text{head}(x, 0)$  is the length of the longest prefix of  $x$  made of zeros;
- $\text{head}(x, 1)$  is the length of the longest prefix of  $x$  made of ones;
- $\text{tail}(x, 0)$  is the length of the longest suffix of  $x$  made of zeros;
- $\text{tail}(x, 1)$  is the length of the longest suffix of  $x$  made of ones;
- $R(x)$  is the reward;
- $R(x) = n$  if  $(\text{head}(x, 0) > t \text{ and } \text{tail}(x, 1) > t) \text{ or } (\text{head}(x, 1) > t \text{ and } \text{tail}(x, 0) > t)$ ;
- $R(x) = 0$  otherwise;
- the threshold  $t$  is a parameter of the function.

This function has six maxima, of which exactly four are global ones.

For example, if  $n = 6$  and  $t = 1$ :

- $f(111111) = 6$  (local maximum)
- $f(111110) = 5$
- $f(111100) = 10$  (global maximum)

Reference:

J. S. De Bonet, C. L. Isbell, and P. Viola. 1996. MIMIC: finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*. Vol. 9. MIT Press, Denver.

Definition at line 128 of file four-peaks.hh.



## 5.97.2 Member Function Documentation

### 5.97.2.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

#### Returns

$2 * \_bv\_size - \_threshold - 1$

Reimplemented from [Function](#).

Definition at line 159 of file four-peaks.hh.

### 5.97.2.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 155 of file four-peaks.hh.

The documentation for this class was generated from the following files:

- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

## 5.98 SpinHerding Class Reference

Herding with spin variables.

```
#include <hnco/algorithms/hea/spin-herding.hh>
```

### Public Types

- enum { [SAMPLE\\_GREEDY](#), [SAMPLE\\_RLS](#), [SAMPLE\\_DLS](#), [LAST\\_SAMPLE](#) }

## Public Member Functions

- [SpinHerd](#)(int n)  
*Constructor.*
- void [init](#)()  
*Initialization.*
- void [sample](#)(const [SpinMoment](#) &target, [bit\\_vector\\_t](#) &x)  
*Sample a bit vector.*
- double [error](#)(const [SpinMoment](#) &target)  
*Compute the error.*

## Getters

- const [SpinMoment](#) & [get\\_delta](#)()  
*Get delta.*

## Setters

- void [set\\_randomize\\_bit\\_order](#)(bool x)  
*Randomize bit order.*
- void [set\\_sampling\\_method](#)(int x)  
*Set the sampling method.*
- void [set\\_num\\_seq\\_updates](#)(int x)  
*Set the number of sequential updates per sample.*
- void [set\\_weight](#)(double x)  
*Set the weight of second order moments.*

## Protected Member Functions

- void [compute\\_delta](#)(const [SpinMoment](#) &target)  
*Compute delta.*
- void [sample\\_greedy](#)([bit\\_vector\\_t](#) &x)  
*Sample by means of a greedy algorithm.*
- double [q\\_derivative](#)(const [bit\\_vector\\_t](#) &x, size\_t i)  
*Derivative of q.*
- double [q\\_variation](#)(const [bit\\_vector\\_t](#) &x, size\_t i)  
*Variation of q.*
- void [sample\\_rls](#)([bit\\_vector\\_t](#) &x)  
*Sample by means of random local search.*
- void [sample\\_dls](#)([bit\\_vector\\_t](#) &x)  
*Sample by means of deterministic local search.*

## Protected Attributes

- [SpinMoment\\_delta](#)  
*Delta moment.*
- [SpinMoment\\_count](#)  
*Counter moment.*
- [permutation\\_t\\_permutation](#)  
*Permutation.*
- `std::uniform_int_distribution< int > _choose_bit`  
*Choose bit.*
- `int _time`  
*Time.*

## Parameters

- `bool _randomize_bit_order = false`  
*Randomize bit order.*
- `int _sampling_method = SAMPLE_GREEDY`  
*Sampling method.*
- `int _num_seq_updates`  
*Number of sequential updates per sample.*
- `double _weight = 1`  
*Weight of second order moments.*

### 5.98.1 Detailed Description

Herding with spin variables.

By spin variables, we mean variables taking values 1 or -1, instead of 0 or 1 in the case of binary variables.

Definition at line 37 of file spin-herding.hh.

### 5.98.2 Member Enumeration Documentation

#### 5.98.2.1 anonymous enum

anonymous enum

#### Enumerator

SAMPLE_GREEDY	Greedy algorithm.
SAMPLE_RLS	Random local search.
SAMPLE_DLS	Deterministic local search.

Definition at line 97 of file spin-herding.hh.

### 5.98.3 Constructor & Destructor Documentation

#### 5.98.3.1 SpinHerdin()

```
SpinHerdin (
    int n ) [inline]
```

Constructor.

##### Parameters

<i>n</i>	Size of bit vectors
----------	---------------------

`_num_seq_updates` is initialized to `n`.

Definition at line 116 of file `spin-herding.hh`.

### 5.98.4 Member Function Documentation

#### 5.98.4.1 q\_variation()

```
double q_variation (
    const bit_vector_t & x,
    size_t i ) [protected]
```

Variation of `q`.

Up to a positive multiplicative constant. Only the sign of the variation matters to local search.

Definition at line 162 of file `spin-herding.cc`.

The documentation for this class was generated from the following files:

- `lib/hnco/algorithms/hea/spin-herding.hh`
- `lib/hnco/algorithms/hea/spin-herding.cc`

## 5.99 SpinMoment Struct Reference

Moment for spin variables.

```
#include <hnco/algorithms/hea/spin-moment.hh>
```

## Public Member Functions

- [SpinMoment](#) (int n)  
*Constructor.*
- void [uniform](#) ()  
*Set the moment to that of the uniform distribution.*
- void [init](#) ()  
*Initialize accumulators.*
- void [add](#) (const [bit\\_vector\\_t](#) &x)  
*Update accumulators.*
- void [average](#) (int count)  
*Compute average.*
- void [update](#) (const [SpinMoment](#) &p, double rate)  
*Update moment.*
- void [bound](#) (double margin)  
*Bound moment.*
- double [distance](#) (const [SpinMoment](#) &p) const  
*Distance.*
- double [norm\\_2](#) () const  
*Compute the norm 2.*
- double [diameter](#) () const  
*Compute the diameter.*
- [size\\_t](#) [size](#) () const  
*Size.*
- void [display](#) (std::ostream &stream)  
*Display.*

## Public Attributes

- std::vector< double > [\\_first](#)  
*First moment.*
- std::vector< std::vector< double > > [\\_second](#)  
*Second moment.*
- double [\\_weight](#) = 1  
*Weight of second order moments.*

### 5.99.1 Detailed Description

Moment for spin variables.

Definition at line 38 of file spin-moment.hh.

### 5.99.2 Member Data Documentation

### 5.99.2.1 `_second`

```
std::vector<std::vector<double> > _second
```

Second moment.

This is a lower triangular matrix with only zeros on the diagonal. Only entries `_second[i][j]` with  $j < i$  are considered.

Definition at line 50 of file `spin-moment.hh`.

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

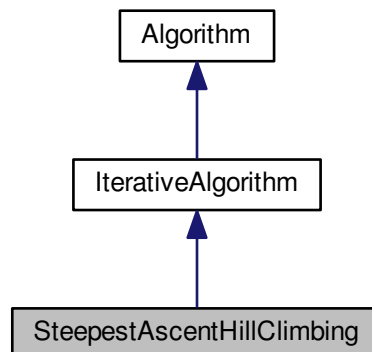
- `lib/hnco/algorithms/hea/spin-moment.hh`
- `lib/hnco/algorithms/hea/spin-moment.cc`

## 5.100 `SteepestAscentHillClimbing` Class Reference

Steepest ascent hill climbing.

```
#include <hnco/algorithms/ls/steepest-ascent-hill-climbing.hh>
```

Inheritance diagram for `SteepestAscentHillClimbing`:



### Public Member Functions

- `SteepestAscentHillClimbing` (int n, `neighborhood::NeighborhoodIterator` \*neighborhood)  
*Constructor.*
- void `init` ()  
*Random initialization.*
- void `init` (const `bit_vector_t` &x)  
*Explicit initialization.*
- void `init` (const `bit_vector_t` &x, double value)  
*Explicit initialization.*

### Protected Member Functions

- void `iterate()`  
*Single iteration.*

### Protected Attributes

- `std::vector< bit_vector_t > _candidates`  
*Potential candidate.*
- `neighborhood::NeighborhoodIterator * _neighborhood`  
*Neighborhood.*

#### 5.100.1 Detailed Description

Steepest ascent hill climbing.

Definition at line 39 of file `steepest-ascent-hill-climbing.hh`.

The documentation for this class was generated from the following files:

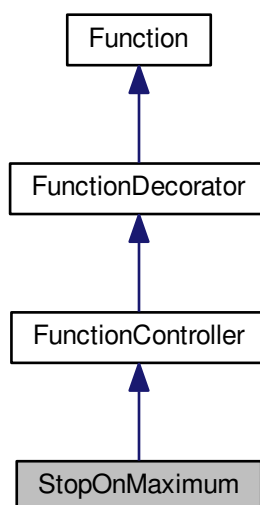
- `lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh`
- `lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc`

## 5.101 StopOnMaximum Class Reference

Stop on maximum.

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for StopOnMaximum:



## Public Member Functions

- [StopOnMaximum](#) ([Function](#) \*function)

*Constructor.*

## Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped↔  
\_bits)  
*Incremental evaluation.*
- void [update](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Update after a safe evaluation.*

## Additional Inherited Members

### 5.101.1 Detailed Description

Stop on maximum.

The member function eval throws an exception MaximumReached when its argument maximizes the decorated function.

#### Warning

The maximum is detected using the equality operator hence the result should be taken with care in case of non integer (floating point) function values.

Definition at line 90 of file function-controller.hh.

### 5.101.2 Constructor & Destructor Documentation

#### 5.101.2.1 StopOnMaximum()

```
StopOnMaximum (
    Function * function ) [inline]
```

Constructor.

#### Parameters

<i>function</i>	Decorated function
-----------------	--------------------



## Precondition

function->[has\\_known\\_maximum\(\)](#)

Definition at line 98 of file function-controller.hh.

### 5.101.3 Member Function Documentation

#### 5.101.3.1 eval()

```
double eval (
    const bit\_vector\_t & x ) [virtual]
```

Evaluate a bit vector.

## Exceptions

<i>MaximumReached</i>	
-----------------------	--

Implements [Function](#).

Definition at line 31 of file function-controller.cc.

#### 5.101.3.2 incremental\_eval()

```
double incremental_eval (
    const bit\_vector\_t & x,
    double value,
    const hnco::sparse\_bit\_vector\_t & flipped_bits ) [virtual]
```

Incremental evaluation.

## Exceptions

<i>MaximumReached</i>	
-----------------------	--

Reimplemented from [Function](#).

Definition at line 43 of file function-controller.cc.

#### 5.101.3.3 update()

```
void update (
    const bit\_vector\_t & x,
    double value ) [virtual]
```

Update after a safe evaluation.

#### Exceptions

<i>MaximumReached</i>	
-----------------------	--

Reimplemented from [Function](#).

Definition at line 55 of file function-controller.cc.

The documentation for this class was generated from the following files:

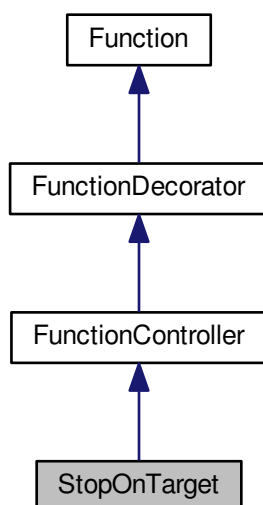
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.102 StopOnTarget Class Reference

Stop on target.

```
#include <hnco/functions/decorators/function-controller.hh>
```

Inheritance diagram for StopOnTarget:



## Public Member Functions

- [StopOnTarget](#) ([Function](#) \*function, double target)  
*Constructor.*

## Evaluation

- double [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- double [incremental\\_eval](#) (const [bit\\_vector\\_t](#) &x, double value, const [hnco::sparse\\_bit\\_vector\\_t](#) &flipped←\_bits)  
*Incremental evaluation.*
- void [update](#) (const [bit\\_vector\\_t](#) &x, double value)  
*Update after a safe evaluation.*

## Private Attributes

- double [\\_target](#)  
*Target.*

## Additional Inherited Members

### 5.102.1 Detailed Description

Stop on target.

The member function eval throws an exception TargetReached when the value of its decorated function reaches a given target.

#### Warning

The target is detected using the greater or equal operator hence the result should be taken with care in case of non integer (floating point) function values.

Definition at line 134 of file function-controller.hh.

### 5.102.2 Constructor & Destructor Documentation

#### 5.102.2.1 StopOnTarget()

```
StopOnTarget (
    Function * function,
    double target ) [inline]
```

Constructor.

## Parameters

<i>function</i>	Decorated function
-----------------	--------------------

Definition at line 144 of file function-controller.hh.

### 5.102.3 Member Function Documentation

#### 5.102.3.1 eval()

```
double eval (
    const bit\_vector\_t & x ) [virtual]
```

Evaluate a bit vector.

## Exceptions

<i>TargetReached</i>	
----------------------	--

Implements [Function](#).

Definition at line 66 of file function-controller.cc.

#### 5.102.3.2 incremental\_eval()

```
double incremental_eval (
    const bit\_vector\_t & x,
    double value,
    const hnco::sparse\_bit\_vector\_t & flipped_bits ) [virtual]
```

Incremental evaluation.

## Exceptions

<i>TargetReached</i>	
----------------------	--

Reimplemented from [Function](#).

Definition at line 76 of file function-controller.cc.

## 5.102.3.3 update()

```
void update (
    const bit\_vector\_t & x,
    double value ) [virtual]
```

Update after a safe evaluation.

## Exceptions

<i>TargetReached</i>	
----------------------	--

Reimplemented from [Function](#).

Definition at line 86 of file function-controller.cc.

The documentation for this class was generated from the following files:

- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.103 Stopwatch Class Reference

Stop watch.

```
#include <hnco/stop-watch.hh>
```

## Public Member Functions

- void [start](#) ()  
*Start.*
- void [stop](#) ()  
*Stop.*
- double [get\\_total](#) ()  
*Get total.*

## Private Attributes

- double [\\_total](#) = 0  
*Total time.*
- clock\_t [\\_start](#)  
*Start time.*

### 5.103.1 Detailed Description

Stop watch.

Definition at line 31 of file stop-watch.hh.

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

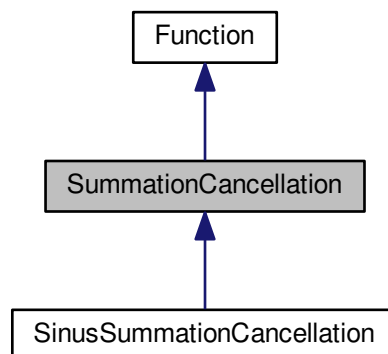
- lib/hnco/stop-watch.hh

## 5.104 SummationCancellation Class Reference

Summation cancellation.

```
#include <hnco/functions/cancellation.hh>
```

Inheritance diagram for SummationCancellation:



### Public Member Functions

- [SummationCancellation](#) (int n)  
*Constructor.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &x)  
*Evaluate a bit vector.*
- [bool has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [double get\\_maximum](#) ()  
*Get the global maximum.*

## Protected Member Functions

- void `convert` (const `bit_vector_t` &x)  
*Convert a bit vector into a real vector.*

## Protected Attributes

- `size_t _bv_size`  
*Bit vector size.*
- `std::vector< double > _buffer`  
*Buffer.*

### 5.104.1 Detailed Description

Summation cancellation.

Encoding of a signed integer:

- bit 0: sign
- bits 1 to 8: two's complement representation

Reference:

S. Baluja and S. Davies. 1997. Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space. Technical Report CMU- CS-97-107. Carnegie-Mellon University.

Definition at line 47 of file `cancellation.hh`.

### 5.104.2 Constructor & Destructor Documentation

#### 5.104.2.1 SummationCancellation()

```
SummationCancellation (
    int n ) [inline]
```

Constructor.

The bit vector size `n` must be a multiple of 9. The size of `_buffer` is then `n / 9`.

Parameters

<code>n</code>	Size of the bit vector
----------------	------------------------

Definition at line 70 of file `cancellation.hh`.

### 5.104.3 Member Function Documentation

#### 5.104.3.1 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 86 of file `cancellation.hh`.

The documentation for this class was generated from the following files:

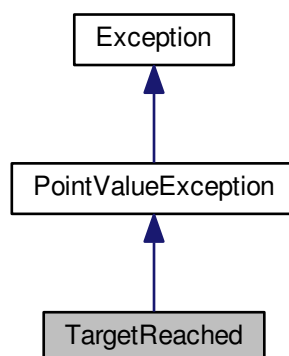
- `lib/hnco/functions/cancellation.hh`
- `lib/hnco/functions/cancellation.cc`

## 5.105 TargetReached Class Reference

target reached

```
#include <hnco/exception.hh>
```

Inheritance diagram for TargetReached:





## Public Member Functions

- [TargetReached](#) (const [point\\_value\\_t](#) &pv)  
*Constructor.*

## Additional Inherited Members

### 5.105.1 Detailed Description

target reached

Definition at line 61 of file exception.hh.

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

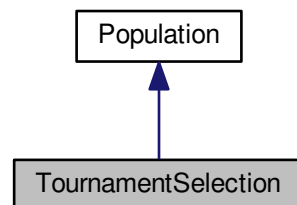
- lib/hnco/exception.hh

## 5.106 TournamentSelection Class Reference

[Population](#) with tournament selection.

```
#include <hnco/algorithms/ea/tournament-selection.hh>
```

Inheritance diagram for TournamentSelection:



## Public Member Functions

- [TournamentSelection](#) (int population\_size, int n)  
*Constructor.*
- const [bit\\_vector\\_t](#) & [select](#) ()  
*Selection.*

## Setters

- void [set\\_tournament\\_size](#) (int x)  
*Set the tournament size.*

## Private Attributes

- `std::uniform_int_distribution< int > _choose_individual`  
*Random index.*

## Parameters

- `int _tournament_size = 10`  
*Tournament size.*

## Additional Inherited Members

### 5.106.1 Detailed Description

[Population](#) with tournament selection.

Definition at line 34 of file tournament-selection.hh.

### 5.106.2 Member Function Documentation

#### 5.106.2.1 `select()`

```
const bit\_vector\_t & select ( )
```

Selection.

The selection only requires that the population be evaluated, not necessarily sorted.

#### Precondition

The population must be evaluated.

Definition at line 33 of file tournament-selection.cc.

The documentation for this class was generated from the following files:

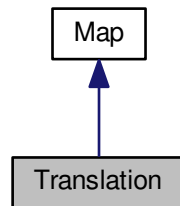
- `lib/hnco/algorithms/ea/tournament-selection.hh`
- `lib/hnco/algorithms/ea/tournament-selection.cc`

## 5.107 Translation Class Reference

Translation.

```
#include <hnco/map.hh>
```

Inheritance diagram for Translation:



### Public Member Functions

- void [random](#) (int n)  
*Random instance.*
- void [map](#) (const [bit\\_vector\\_t](#) &input, [bit\\_vector\\_t](#) &output)  
*Map.*
- size\_t [get\\_input\\_size](#) ()  
*Get input size.*
- size\_t [get\\_output\\_size](#) ()  
*Get output size.*
- bool [is\\_surjective](#) ()  
*Check for surjective map.*

### Private Member Functions

- template<class Archive >  
void [save](#) (Archive &ar, const unsigned int version) const  
*Save.*
- template<class Archive >  
void [load](#) (Archive &ar, const unsigned int version)  
*Load.*

### Private Attributes

- [bit\\_vector\\_t \\_bv](#)  
*Translation vector.*

## Friends

- class **boost::serialization::access**

### 5.107.1 Detailed Description

#### [Translation.](#)

A translation is an affine map  $f$  from  $Z_2^n$  to itself defined by  $f(x) = x + b$ , where  $b$  is an  $n$ -dimensional bit vector.

Definition at line 70 of file map.hh.

### 5.107.2 Member Function Documentation

#### 5.107.2.1 `is_surjective()`

```
bool is_surjective ( ) [inline], [virtual]
```

Check for surjective map.

#### Returns

true

Reimplemented from [Map](#).

Definition at line 121 of file map.hh.

The documentation for this class was generated from the following files:

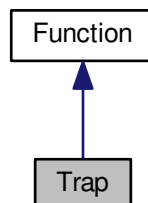
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.108 Trap Class Reference

#### [Trap.](#)

```
#include <hnco/functions/trap.hh>
```

Inheritance diagram for Trap:



## Public Member Functions

- [Trap](#) (int bv\_size, int num\_traps)  
*Constructor.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*
- [bool has\\_known\\_maximum](#) ()  
*Check for a known maximum.*
- [double get\\_maximum](#) ()  
*Get the global maximum.*

## Private Attributes

- [size\\_t \\_bv\\_size](#)  
*Bit vector size.*
- [int \\_num\\_traps](#)  
*Number of traps.*
- [int \\_trap\\_size](#)  
*Trap size.*

### 5.108.1 Detailed Description

[Trap](#).

Reference:

Kalyanmoy Deb and David E. Goldberg. 1993. Analyzing Deception in Trap Functions. In Foundations of Genetic Algorithms 2, L. Darrell Whitley (Ed.). Morgan Kaufmann, San Mateo, CA, 93–108.

Definition at line 43 of file trap.hh.

### 5.108.2 Constructor & Destructor Documentation

#### 5.108.2.1 [Trap\(\)](#)

```
Trap (
    int bv_size,
    int num_traps ) [inline]
```

Constructor.

Parameters

<i>bv_size</i>	Bit vector size
<i>num_traps</i>	Number of traps

**Warning**

`bv_size` must be a multiple of `num_traps`

Definition at line 64 of file `trap.hh`.

### 5.108.3 Member Function Documentation

#### 5.108.3.1 `get_maximum()`

```
double get_maximum ( ) [inline], [virtual]
```

Get the global maximum.

**Returns**

`_bv_size`

Reimplemented from [Function](#).

Definition at line 88 of file `trap.hh`.

#### 5.108.3.2 `has_known_maximum()`

```
bool has_known_maximum ( ) [inline], [virtual]
```

Check for a known maximum.

**Returns**

`true`

Reimplemented from [Function](#).

Definition at line 84 of file `trap.hh`.

The documentation for this class was generated from the following files:

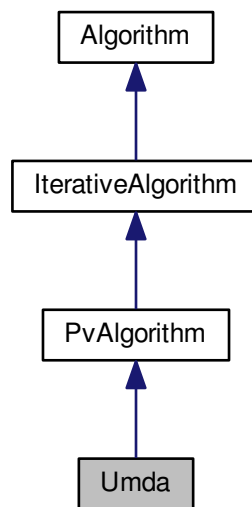
- `lib/hnco/functions/trap.hh`
- `lib/hnco/functions/trap.cc`

## 5.109 Umda Class Reference

Univariate marginal distribution algorithm.

```
#include <hnco/algorithms/pv/umda.hh>
```

Inheritance diagram for Umda:



### Public Member Functions

- [Umda](#) (int n, int population\_size)  
*Constructor.*
- void [init](#) ()  
*Initialization.*

### Setters

- void [set\\_selection\\_size](#) (int x)  
*Set the selection size.*

### Protected Member Functions

- void [iterate](#) ()  
*Single iteration.*

## Protected Attributes

- [Population \\_population](#)  
*Population.*
- [pv\\_t \\_mean](#)  
*Mean of selected bit vectors.*

## Parameters

- `int _selection_size = 1`  
*Selection size.*

## Additional Inherited Members

### 5.109.1 Detailed Description

Univariate marginal distribution algorithm.

Reference:

H. Mühlenbein. 1997. The equation for response to selection and its use for prediction. *Evolutionary Computation* 5, 3 (1997), 303–346.

Definition at line 40 of file `umda.hh`.

The documentation for this class was generated from the following files:

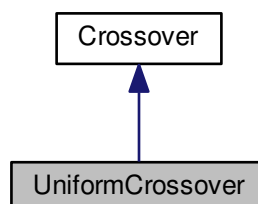
- `lib/hnco/algorithms/pv/umda.hh`
- `lib/hnco/algorithms/pv/umda.cc`

## 5.110 UniformCrossover Class Reference

Uniform crossover.

```
#include <hnco/algorithms/ea/crossover.hh>
```

Inheritance diagram for `UniformCrossover`:





## Public Member Functions

- void [breed](#) (const [bit\\_vector\\_t](#) &parent1, const [bit\\_vector\\_t](#) &parent2, [bit\\_vector\\_t](#) &offspring)  
*Breed.*

### 5.110.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

### 5.110.2 Member Function Documentation

#### 5.110.2.1 breed()

```
void breed (  
    const bit\_vector\_t & parent1,  
    const bit\_vector\_t & parent2,  
    bit\_vector\_t & offspring ) [virtual]
```

Breed.

The offspring is the uniform crossover of two parents.

#### Parameters

<i>parent1</i>	First parent
<i>parent2</i>	Second parent
<i>offspring</i>	Offspring

Implements [Crossover](#).

Definition at line 30 of file crossover.cc.

The documentation for this class was generated from the following files:

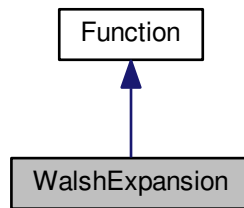
- lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

## 5.111 WalshExpansion Class Reference

Walsh expansion.

```
#include <hnco/functions/walsh/walsh-expansion.hh>
```

Inheritance diagram for WalshExpansion:



### Public Member Functions

- [WalshExpansion](#) ()  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `void` [random](#) (int n, int num\_features, double stddev)  
*Random instance.*
- `double` [eval](#) (const `bit_vector_t` &)  
*Evaluate a bit vector.*
- `void` [display](#) (std::ostream &stream)  
*Display.*
- `void` [set\\_terms](#) (const std::vector< [Function::WalshTransformTerm](#) > terms)  
*Set terms.*

### Private Member Functions

- `template<class Archive >`  
`void` [serialize](#) (Archive &ar, const unsigned int version)  
*Save.*

### Private Attributes

- `std::vector< Function::WalshTransformTerm >` `_terms`  
*Terms.*

### Friends

- `class` `boost::serialization::access`

### 5.111.1 Detailed Description

Walsh expansion.

Its expression is of the form

$$f(x) = \sum_u a_u (-1)^{x \cdot u}$$

where the sum is over a subset of  $\{0, 1\}^n$  and  $x \cdot u = \sum_i x_i u_i$  is mod 2. The real numbers  $a_u$  are the coefficients of the expansion and the bit vectors  $u$  are its feature vectors.

Definition at line 53 of file walsh-expansion.hh.

### 5.111.2 Member Function Documentation

#### 5.111.2.1 random()

```
void random (
    int n,
    int num_features,
    double stddev )
```

Random instance.

#### Parameters

<i>n</i>	Size of bit vector
<i>num_features</i>	Number of feature vectors
<i>stddev</i>	Standard deviation of the coefficients

Definition at line 34 of file walsh-expansion.cc.

The documentation for this class was generated from the following files:

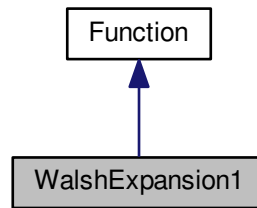
- lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

## 5.112 WalshExpansion1 Class Reference

Walsh expansion of degree 1.

```
#include <hnco/functions/walsh/walsh-expansion-1.hh>
```

Inheritance diagram for WalshExpansion1:



### Public Member Functions

- [WalshExpansion1](#) ()  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `void` [random](#) (int n, double stddev)  
*Random instance.*
- `double` [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Private Member Functions

- `template<class Archive >`  
`void` [serialize](#) (Archive &ar, const unsigned int version)  
*Serialize.*

### Private Attributes

- `std::vector< double >` [\\_linear](#)  
*Linear part.*

### Friends

- `class` **`boost::serialization::access`**

### 5.112.1 Detailed Description

Walsh expansion of degree 1.

Its expression is of the form

$$f(x) = \sum_i a_i (1 - 2x_i)$$

or equivalently

$$f(x) = \sum_i a_i (-1)^{x_i}$$

Definition at line 50 of file `walsh-expansion-1.hh`.

### 5.112.2 Member Function Documentation

#### 5.112.2.1 random()

```
void random (
    int n,
    double stddev )
```

Random instance.

##### Parameters

<i>n</i>	Size of bit vector
<i>stddev</i>	Standard deviation of the coefficients

Definition at line 33 of file walsh-expansion-1.cc.

The documentation for this class was generated from the following files:

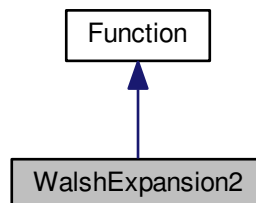
- lib/hnco/functions/walsh/walsh-expansion-1.hh
- lib/hnco/functions/walsh/walsh-expansion-1.cc

## 5.113 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

```
#include <hnco/functions/walsh/walsh-expansion-2.hh>
```

Inheritance diagram for WalshExpansion2:



## Public Member Functions

- [WalshExpansion2](#) ()  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `void` [random](#) (int n, double stddev\_lin, double stddev\_quad)  
*Random instance.*
- `double` [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Private Member Functions

- `template<class Archive >`  
`void` [serialize](#) (Archive &ar, const unsigned int version)  
*Serialize.*

## Private Attributes

- `std::vector< double >` [\\_linear](#)  
*Linear part.*
- `std::vector< std::vector< double > >` [\\_quadratic](#)  
*Quadratic part.*

## Friends

- `class` **boost::serialization::access**

### 5.113.1 Detailed Description

Walsh expansion of degree 2.

Its expression is of the form

$$f(x) = \sum_i a_i (1 - 2x_i) + \sum_{i < j} a_{ij} (1 - 2x_i)(1 - 2x_j)$$

or equivalently

$$f(x) = \sum_i a_i (-1)^{x_i} + \sum_{i < j} a_{ij} (-1)^{x_i + x_j}$$

where the sum  $x_i + x_j$  is mod 2 (xor).

Definition at line 52 of file walsh-expansion-2.hh.

### 5.113.2 Member Function Documentation

#### 5.113.2.1 random()

```
void random (
    int n,
    double stddev_lin,
    double stddev_quad )
```

Random instance.

## Parameters

<i>n</i>	Size of bit vector
<i>stddev_lin</i>	Standard deviation of the coefficients of the linear part
<i>stddev_quad</i>	Standard deviation of the coefficients of the quadratic part

Definition at line 33 of file walsh-expansion-2.cc.

### 5.113.3 Member Data Documentation

#### 5.113.3.1 `_quadratic`

```
std::vector<std::vector<double> > _quadratic [private]
```

Quadratic part.

Represented as a lower triangular matrix (without its diagonal).

Definition at line 75 of file walsh-expansion-2.hh.

The documentation for this class was generated from the following files:

- lib/hnco/functions/walsh/walsh-expansion-2.hh
- lib/hnco/functions/walsh/walsh-expansion-2.cc

## 5.114 Function::WalshTransformTerm Struct Reference

Walsh transform term.

```
#include <hnco/functions/function.hh>
```

### Public Member Functions

- `template<class Archive >`  
void [serialize](#) (Archive &ar, const unsigned int version)  
*Serialize.*

### Public Attributes

- `std::vector< bool >` [feature](#)  
*Feature.*
- `double` [coefficient](#)  
*Coefficient.*

### 5.114.1 Detailed Description

Walsh transform term.

Definition at line 45 of file function.hh.

### 5.114.2 Member Data Documentation

#### 5.114.2.1 feature

```
std::vector<bool> feature
```

Feature.

Implemented with a vector bool instead of a bit\_vector\_t to reduce the memory consumption.

Definition at line 52 of file function.hh.

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

- lib/hnco/functions/function.hh



# Index

- `_compare_index_value`
    - `hnco::algorithm::Population`, 176
  - `_expression`
    - `hnco::function::AbstractMaxSat`, 33
  - `_functions`
    - `hnco::algorithm::Algorithm`, 41
  - `_lookup`
    - `hnco::algorithm::Population`, 176
  - `_q`
    - `hnco::function::Qubo`, 189
  - `_quadratic`
    - `hnco::function::WalshExpansion2`, 235
  - `_second`
    - `hnco::algorithm::hea::SpinMoment`, 209
- AbstractLabs, 31
- AbstractMaxSat, 32
- AdditiveGaussianNoise, 34
- AffineMap, 36
- Algorithm, 38
- 
- `bernoulli_trials`
  - `hnco::neighborhood::MultiBitFlip`, 136
- BernoulliProcess, 41
  - `hnco::neighborhood::BernoulliProcess`, 42, 43
- BiasedCrossover, 44
- `bit_t`
  - `hnco`, 17
- BitHerding, 45
- BitMoment, 47
- `bm_add_rows`
  - `hnco`, 17
- `bm_identity`
  - `hnco`, 18
- `bm_invert`
  - `hnco`, 18
- `bm_multiply`
  - `hnco`, 19
- `bm_rank`
  - `hnco`, 19
- `bm_row_echelon_form`
  - `hnco`, 19
- `bm_solve`
  - `hnco`, 19
- `bm_solve_upper_triangular`
  - `hnco`, 20
- BmPbil, 48
- breed
  - `hnco::algorithm::BiasedCrossover`, 44
  - `hnco::algorithm::Crossover`, 59
  - `hnco::algorithm::UniformCrossover`, 229
- `bv_from_vector_bool`
  - `hnco`, 21
- `bv_to_vector_bool`
  - `hnco`, 21
- 
- Cache, 52
  - `hnco::function::Cache`, 53
- CallCounter, 54
- `comma_selection`
  - `hnco::algorithm::Population`, 172
- CompactGa, 55
- CompleteSearch, 57
- `compute_walsh_transform`
  - `hnco::function::Function`, 72
- Crossover, 58
- 
- DeceptiveJump, 59
- 
- EqualProducts, 61
- Error, 63
- eval
  - `hnco::function::OnBudgetFunction`, 154
  - `hnco::function::ProgressTracker`, 181
  - `hnco::function::StopOnMaximum`, 213
  - `hnco::function::StopOnTarget`, 216
- Exception, 64
- 
- Factorization, 65
  - `hnco::function::Factorization`, 66
- feature
  - `hnco::function::Function::WalshTransformTerm`, 236
- FirstAscentHillClimbing, 67
- FourPeaks, 68
- Function, 71
- `Function::WalshTransformTerm`, 235
- FunctionController, 75
- FunctionDecorator, 76
- FunctionMapComposition, 77
  - `hnco::function::FunctionMapComposition`, 78
- FunctionModifier, 80
- FunctionPlugin, 81
  - `hnco::function::FunctionPlugin`, 82
- 
- GeneticAlgorithm, 82
  - `hnco::algorithm::GeneticAlgorithm`, 84
- `get_best_bv`
  - `hnco::algorithm::Population`, 172, 173
- `get_best_value`
  - `hnco::algorithm::Population`, 174

- get\_last\_improvement
  - hnco::function::ProgressTracker, 181
- get\_maximum
  - hnco::function::AdditiveGaussianNoise, 35
  - hnco::function::DeceptiveJump, 60
  - hnco::function::FourPeaks, 70
  - hnco::function::Function, 73
  - hnco::function::FunctionMapComposition, 79
  - hnco::function::Hiff, 97
  - hnco::function::Jump, 106
  - hnco::function::LeadingOnes, 110
  - hnco::function::LongPath, 118
  - hnco::function::Needle, 141
  - hnco::function::Negation, 143
  - hnco::function::OneMax, 156
  - hnco::function::Plateau, 168
  - hnco::function::PriorNoise, 178
  - hnco::function::Ridge, 197
  - hnco::function::SixPeaks, 205
  - hnco::function::Trap, 226
- get\_worst\_bv
  - hnco::algorithm::Population, 174, 175
- HammingBall, 85
  - hnco::neighborhood::HammingBall, 86
- HammingSphere, 87
  - hnco::neighborhood::HammingSphere, 88
- HammingSphereIterator, 88
  - hnco::neighborhood::HammingSphereIterator, 90
- has\_known\_maximum
  - hnco::function::AdditiveGaussianNoise, 35
  - hnco::function::DeceptiveJump, 60
  - hnco::function::FourPeaks, 70
  - hnco::function::FunctionMapComposition, 79
  - hnco::function::Hiff, 97
  - hnco::function::Jump, 106
  - hnco::function::LeadingOnes, 110
  - hnco::function::LinearFunction, 112
  - hnco::function::LongPath, 119
  - hnco::function::Needle, 141
  - hnco::function::Negation, 143
  - hnco::function::OneMax, 157
  - hnco::function::Plateau, 168
  - hnco::function::PriorNoise, 178
  - hnco::function::Ridge, 197
  - hnco::function::SixPeaks, 205
  - hnco::function::SummationCancellation, 220
  - hnco::function::Trap, 226
- Hboa, 90
- Hea
  - hnco::algorithm::hea::Hea, 94
- Hea< Moment, Herding >, 91
- Hiff, 96
- hnco, 13
  - bit\_t, 17
  - bm\_add\_rows, 17
  - bm\_identity, 18
  - bm\_invert, 18
  - bm\_multiply, 19
  - bm\_rank, 19
  - bm\_row\_echelon\_form, 19
  - bm\_solve, 19
  - bm\_solve\_upper\_triangular, 20
  - bv\_from\_vector\_bool, 21
  - bv\_to\_vector\_bool, 21
  - perm\_identity, 21
  - perm\_random, 21
  - sbm\_multiply, 22
  - sparse\_bit\_matrix\_t, 17
  - sparse\_bit\_vector\_t, 17
- hnco::AffineMap
  - is\_surjective, 37
  - random, 37
- hnco::Injection
  - Injection, 100
- hnco::LinearMap
  - is\_surjective, 114
  - random, 114
- hnco::Map
  - is\_surjective, 121
- hnco::MapComposition
  - is\_surjective, 123
  - MapComposition, 123
- hnco::Permutation
  - is\_surjective, 166
- hnco::Projection
  - is\_surjective, 185
  - Projection, 184
- hnco::Translation
  - is\_surjective, 224
- hnco::algorithm, 22
- hnco::algorithm::Algorithm
  - \_functions, 41
  - set\_solution, 40
  - update\_solution, 40
- hnco::algorithm::BiasedCrossover
  - breed, 44
- hnco::algorithm::Crossover
  - breed, 59
- hnco::algorithm::GeneticAlgorithm
  - GeneticAlgorithm, 84
  - set\_allow\_stay, 85
- hnco::algorithm::IterativeAlgorithm
  - IterativeAlgorithm, 102
  - maximize, 103
  - set\_num\_iterations, 103
- hnco::algorithm::MuCommaLambdaEa
  - MuCommaLambdaEa, 134
  - set\_allow\_stay, 135
- hnco::algorithm::MuPlusLambdaEa
  - MuPlusLambdaEa, 139
  - set\_allow\_stay, 139
- hnco::algorithm::OnePlusLambdaCommaLambdaGa
  - OnePlusLambdaCommaLambdaGa, 159
- hnco::algorithm::OnePlusOneEa
  - OnePlusOneEa, 161
  - set\_allow\_stay, 162

- set\_num\_iterations, [162](#)
- hnco::algorithm::Population
  - \_compare\_index\_value, [176](#)
  - \_lookup, [176](#)
  - comma\_selection, [172](#)
  - get\_best\_bv, [172](#), [173](#)
  - get\_best\_value, [174](#)
  - get\_worst\_bv, [174](#), [175](#)
  - plus\_selection, [175](#)
- hnco::algorithm::RandomLocalSearch
  - set\_patience, [192](#)
- hnco::algorithm::SimulatedAnnealing
  - init\_beta, [199](#)
- hnco::algorithm::TournamentSelection
  - select, [222](#)
- hnco::algorithm::UniformCrossover
  - breed, [229](#)
- hnco::algorithm::bm\_pbil, [25](#)
- hnco::algorithm::bm\_pbil::BmPbil
  - set\_selection\_size, [51](#)
- hnco::algorithm::eda, [25](#)
- hnco::algorithm::hea, [26](#)
- hnco::algorithm::hea::Hea
  - Hea, [94](#)
  - set\_reset\_period, [95](#)
  - set\_selection\_size, [95](#)
- hnco::algorithm::hea::SpinHerdng
  - q\_variation, [208](#)
  - SpinHerdng, [208](#)
- hnco::algorithm::hea::SpinMoment
  - \_second, [209](#)
- hnco::exception, [26](#)
- hnco::function, [27](#)
- hnco::function::AbstractMaxSat
  - \_expression, [33](#)
  - load, [33](#)
- hnco::function::AdditiveGaussianNoise
  - get\_maximum, [35](#)
  - has\_known\_maximum, [35](#)
- hnco::function::Cache
  - Cache, [53](#)
  - provides\_incremental\_evaluation, [53](#)
- hnco::function::DeceptiveJump
  - get\_maximum, [60](#)
  - has\_known\_maximum, [60](#)
- hnco::function::EqualProducts
  - random, [62](#)
- hnco::function::Factorization
  - Factorization, [66](#)
- hnco::function::FourPeaks
  - get\_maximum, [70](#)
  - has\_known\_maximum, [70](#)
- hnco::function::Function
  - compute\_walsh\_transform, [72](#)
  - get\_maximum, [73](#)
  - incremental\_eval, [73](#)
  - provides\_incremental\_evaluation, [74](#)
  - safe\_eval, [74](#)
- hnco::function::Function::WalshTransformTerm
  - feature, [236](#)
- hnco::function::FunctionController
  - provides\_incremental\_evaluation, [76](#)
- hnco::function::FunctionMapComposition
  - FunctionMapComposition, [78](#)
  - get\_maximum, [79](#)
  - has\_known\_maximum, [79](#)
- hnco::function::FunctionPlugin
  - FunctionPlugin, [82](#)
- hnco::function::Hiff
  - get\_maximum, [97](#)
  - has\_known\_maximum, [97](#)
- hnco::function::Jump
  - get\_maximum, [106](#)
  - has\_known\_maximum, [106](#)
- hnco::function::LeadingOnes
  - get\_maximum, [110](#)
  - has\_known\_maximum, [110](#)
- hnco::function::LinearFunction
  - has\_known\_maximum, [112](#)
  - random, [112](#)
- hnco::function::LongPath
  - get\_maximum, [118](#)
  - has\_known\_maximum, [119](#)
- hnco::function::MaxNae3Sat
  - load, [125](#)
- hnco::function::MaxSat
  - random, [127](#)
- hnco::function::Needle
  - get\_maximum, [141](#)
  - has\_known\_maximum, [141](#)
- hnco::function::Negation
  - get\_maximum, [143](#)
  - has\_known\_maximum, [143](#)
  - provides\_incremental\_evaluation, [143](#)
- hnco::function::NkLandscape
  - random, [150](#)
- hnco::function::OnBudgetFunction
  - eval, [154](#)
  - incremental\_eval, [154](#)
  - update, [154](#)
- hnco::function::OneMax
  - get\_maximum, [156](#)
  - has\_known\_maximum, [157](#)
  - provides\_incremental\_evaluation, [157](#)
- hnco::function::Plateau
  - get\_maximum, [168](#)
  - has\_known\_maximum, [168](#)
- hnco::function::PriorNoise
  - get\_maximum, [178](#)
  - has\_known\_maximum, [178](#)
  - provides\_incremental\_evaluation, [178](#)
- hnco::function::ProgressTracker
  - eval, [181](#)
  - get\_last\_improvement, [181](#)
  - incremental\_eval, [181](#)
  - update, [182](#)

- hnco::function::Qubo
  - \_q, 189
  - load, 189
- hnco::function::Ridge
  - get\_maximum, 197
  - has\_known\_maximum, 197
- hnco::function::SixPeaks
  - get\_maximum, 205
  - has\_known\_maximum, 205
- hnco::function::StopOnMaximum
  - eval, 213
  - incremental\_eval, 213
  - StopOnMaximum, 212
  - update, 213
- hnco::function::StopOnTarget
  - eval, 216
  - incremental\_eval, 216
  - StopOnTarget, 215
  - update, 216
- hnco::function::SummationCancellation
  - has\_known\_maximum, 220
  - SummationCancellation, 219
- hnco::function::Trap
  - get\_maximum, 226
  - has\_known\_maximum, 226
  - Trap, 225
- hnco::function::WalshExpansion
  - random, 231
- hnco::function::WalshExpansion1
  - random, 233
- hnco::function::WalshExpansion2
  - \_quadratic, 235
  - random, 234
- hnco::neighborhood, 29
- hnco::neighborhood::BernoulliProcess
  - BernoulliProcess, 42, 43
  - set\_allow\_stay, 43
  - set\_probability, 43
- hnco::neighborhood::HammingBall
  - HammingBall, 86
- hnco::neighborhood::HammingSphere
  - HammingSphere, 88
- hnco::neighborhood::HammingSphereIterator
  - HammingSphereIterator, 90
- hnco::neighborhood::MultiBitFlip
  - bernoulli\_trials, 136
  - MultiBitFlip, 136
  - reservoir\_sampling, 137
- hnco::neighborhood::Neighborhood
  - map, 146
  - mutate, 147
  - Neighborhood, 146
- hnco::neighborhood::NeighborhoodIterator
  - NeighborhoodIterator, 148
- hnco::neighborhood::SingleBitFlipIterator
  - SingleBitFlipIterator, 202
- hnco::random, 29
- HncoEvaluator, 97
- HypercubeIterator, 98
- incremental\_eval
  - hnco::function::Function, 73
  - hnco::function::OnBudgetFunction, 154
  - hnco::function::ProgressTracker, 181
  - hnco::function::StopOnMaximum, 213
  - hnco::function::StopOnTarget, 216
- init\_beta
  - hnco::algorithm::SimulatedAnnealing, 199
- Injection, 99
  - hnco::Injection, 100
- is\_surjective
  - hnco::AffineMap, 37
  - hnco::LinearMap, 114
  - hnco::Map, 121
  - hnco::MapComposition, 123
  - hnco::Permutation, 166
  - hnco::Projection, 185
  - hnco::Translation, 224
- IterativeAlgorithm, 101
  - hnco::algorithm::IterativeAlgorithm, 102
- Iterator, 104
- Jump, 105
- Labs, 107
- LabsMeritFactor, 108
- LastEvaluation, 109
- LeadingOnes, 109
- LinearFunction, 111
- LinearMap, 113
- load
  - hnco::function::AbstractMaxSat, 33
  - hnco::function::MaxNae3Sat, 125
  - hnco::function::Qubo, 189
- LocalMaximum, 115
- LogContext, 116
- LongPath, 117
- Ltga, 119
- Map, 120
- map
  - hnco::neighborhood::Neighborhood, 146
- MapComposition, 122
  - hnco::MapComposition, 123
- MaxNae3Sat, 125
- MaxSat, 126
- maximize
  - hnco::algorithm::IterativeAlgorithm, 103
- MaximumReached, 124
- Mimic, 128
- Mmas, 130
- Model, 131
- ModelParameters, 132
- MuCommaLambdaEa, 133
  - hnco::algorithm::MuCommaLambdaEa, 134
- MuPlusLambdaEa, 137
  - hnco::algorithm::MuPlusLambdaEa, 139

- MultiBitFlip, 135
  - hnco::neighborhood::MultiBitFlip, 136
- mutate
  - hnco::neighborhood::Neighborhood, 147
- Needle, 140
- Negation, 142
- Neighborhood, 144
  - hnco::neighborhood::Neighborhood, 146
- NeighborhoodIterator, 147
  - hnco::neighborhood::NeighborhoodIterator, 148
- NkLandscape, 148
- NpsPbil, 150
- OnBudgetFunction, 152
- OneMax, 155
- OnePlusLambdaCommaLambdaGa, 158
  - hnco::algorithm::OnePlusLambdaCommaLambdaGa, 159
- OnePlusOneEa, 160
  - hnco::algorithm::OnePlusOneEa, 161
- ParameterLessPopulationPyramid, 162
- Pbil, 164
- perm\_identity
  - hnco, 21
- perm\_random
  - hnco, 21
- Permutation, 165
- Plateau, 167
- plus\_selection
  - hnco::algorithm::Population, 175
- PointValueException, 169
- Population, 170
- PriorNoise, 177
- ProgressTracker, 179
- ProgressTracker::Event, 64
- ProgressTrackerContext, 182
- Projection, 183
  - hnco::Projection, 184
- provides\_incremental\_evaluation
  - hnco::function::Cache, 53
  - hnco::function::Function, 74
  - hnco::function::FunctionController, 76
  - hnco::function::Negation, 143
  - hnco::function::OneMax, 157
  - hnco::function::PriorNoise, 178
- PvAlgorithm, 185
- q\_variation
  - hnco::algorithm::hea::SpinHerding, 208
- Qubo, 187
- Random, 190
- random
  - hnco::AffineMap, 37
  - hnco::LinearMap, 114
  - hnco::function::EqualProducts, 62
  - hnco::function::LinearFunction, 112
  - hnco::function::MaxSat, 127
  - hnco::function::NkLandscape, 150
  - hnco::function::WalshExpansion, 231
  - hnco::function::WalshExpansion1, 233
  - hnco::function::WalshExpansion2, 234
- RandomLocalSearch, 190
- RandomSearch, 192
- RandomWalk, 193
- reservoir\_sampling
  - hnco::neighborhood::MultiBitFlip, 137
- Restart, 195
- Ridge, 196
- safe\_eval
  - hnco::function::Function, 74
- sbm\_multiply
  - hnco, 22
- select
  - hnco::algorithm::TournamentSelection, 222
- set\_allow\_stay
  - hnco::algorithm::GeneticAlgorithm, 85
  - hnco::algorithm::MuCommaLambdaEa, 135
  - hnco::algorithm::MuPlusLambdaEa, 139
  - hnco::algorithm::OnePlusOneEa, 162
  - hnco::neighborhood::BernoulliProcess, 43
- set\_num\_iterations
  - hnco::algorithm::IterativeAlgorithm, 103
  - hnco::algorithm::OnePlusOneEa, 162
- set\_patience
  - hnco::algorithm::RandomLocalSearch, 192
- set\_probability
  - hnco::neighborhood::BernoulliProcess, 43
- set\_reset\_period
  - hnco::algorithm::hea::Hea, 95
- set\_selection\_size
  - hnco::algorithm::bm\_pbil::BmPbil, 51
  - hnco::algorithm::hea::Hea, 95
- set\_solution
  - hnco::algorithm::Algorithm, 40
- SimulatedAnnealing, 198
- SingleBitFlip, 200
- SingleBitFlipIterator, 201
  - hnco::neighborhood::SingleBitFlipIterator, 202
- SinusSummationCancellation, 202
- SixPeaks, 203
- sparse\_bit\_matrix\_t
  - hnco, 17
- sparse\_bit\_vector\_t
  - hnco, 17
- SpinHerding, 205
  - hnco::algorithm::hea::SpinHerding, 208
- SpinMoment, 208
- SteepestAscentHillClimbing, 210
- StopOnMaximum, 211
  - hnco::function::StopOnMaximum, 212
- StopOnTarget, 214
  - hnco::function::StopOnTarget, 215
- StopWatch, 217
- SummationCancellation, 218

- hnco::function::SummationCancellation, [219](#)
- TargetReached, [220](#)
- TournamentSelection, [221](#)
- Translation, [223](#)
- Trap, [224](#)
  - hnco::function::Trap, [225](#)
- Umda, [227](#)
- UniformCrossover, [228](#)
- update
  - hnco::function::OnBudgetFunction, [154](#)
  - hnco::function::ProgressTracker, [182](#)
  - hnco::function::StopOnMaximum, [213](#)
  - hnco::function::StopOnTarget, [216](#)
- update\_solution
  - hnco::algorithm::Algorithm, [40](#)
- WalshExpansion, [229](#)
- WalshExpansion1, [231](#)
- WalshExpansion2, [233](#)