

HNCO

0.9

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>11</b>
4.1	hnco Namespace Reference . . . . .	11
4.1.1	Detailed Description . . . . .	14
4.1.2	Typedef Documentation . . . . .	14
4.1.2.1	bit_t . . . . .	14
4.1.2.2	sparse_bit_matrix_t . . . . .	15
4.1.2.3	sparse_bit_vector_t . . . . .	15
4.1.3	Function Documentation . . . . .	15
4.1.3.1	bm_add_rows() . . . . .	15
4.1.3.2	bm_identity() . . . . .	16
4.1.3.3	bm_invert() . . . . .	16
4.1.3.4	bm_multiply() . . . . .	17
4.1.3.5	bm_rank() . . . . .	17
4.1.3.6	bm_row_echelon_form() . . . . .	17
4.1.3.7	bm_solve() . . . . .	17
4.1.3.8	bm_solve_upper_triangular() . . . . .	18

4.1.3.9	<a href="#">bv_from_vector_bool()</a>	19
4.1.3.10	<a href="#">bv_to_vector_bool()</a>	19
4.1.3.11	<a href="#">sbm_multiply()</a>	19
4.2	<a href="#">hnco::algorithm Namespace Reference</a>	20
4.2.1	<a href="#">Detailed Description</a>	22
4.3	<a href="#">hnco::algorithm::bm_pbil Namespace Reference</a>	22
4.3.1	<a href="#">Detailed Description</a>	22
4.4	<a href="#">hnco::algorithm::hea Namespace Reference</a>	22
4.4.1	<a href="#">Detailed Description</a>	23
4.5	<a href="#">hnco::exception Namespace Reference</a>	23
4.5.1	<a href="#">Detailed Description</a>	23
4.6	<a href="#">hnco::function Namespace Reference</a>	23
4.6.1	<a href="#">Detailed Description</a>	25
4.7	<a href="#">hnco::neighborhood Namespace Reference</a>	25
4.7.1	<a href="#">Detailed Description</a>	26
4.8	<a href="#">hnco::random Namespace Reference</a>	26
4.8.1	<a href="#">Detailed Description</a>	26
<b>5</b>	<b><a href="#">Class Documentation</a></b>	<b>27</b>
5.1	<a href="#">AdditiveGaussianNoise Class Reference</a>	27
5.1.1	<a href="#">Detailed Description</a>	28
5.1.2	<a href="#">Member Function Documentation</a>	28
5.1.2.1	<a href="#">get_maximum()</a>	28
5.1.2.2	<a href="#">has_known_maximum()</a>	28
5.2	<a href="#">AffineMap Class Reference</a>	29
5.2.1	<a href="#">Detailed Description</a>	30
5.2.2	<a href="#">Member Function Documentation</a>	30
5.2.2.1	<a href="#">is_surjective()</a>	30
5.2.2.2	<a href="#">random()</a>	30
5.3	<a href="#">Algorithm Class Reference</a>	31
5.3.1	<a href="#">Detailed Description</a>	33

5.3.2	Member Data Documentation . . . . .	33
5.3.2.1	_functions . . . . .	33
5.4	BernoulliProcess Class Reference . . . . .	33
5.4.1	Detailed Description . . . . .	34
5.4.2	Constructor & Destructor Documentation . . . . .	34
5.4.2.1	BernoulliProcess() [1/2] . . . . .	34
5.4.2.2	BernoulliProcess() [2/2] . . . . .	35
5.4.3	Member Function Documentation . . . . .	35
5.4.3.1	set_allow_stay() . . . . .	35
5.4.3.2	set_probability() . . . . .	35
5.5	BiasedCrossover Class Reference . . . . .	36
5.5.1	Detailed Description . . . . .	36
5.5.2	Member Function Documentation . . . . .	36
5.5.2.1	breed() . . . . .	36
5.6	BinaryHerding Class Reference . . . . .	37
5.6.1	Detailed Description . . . . .	38
5.6.2	Member Enumeration Documentation . . . . .	38
5.6.2.1	anonymous enum . . . . .	38
5.7	BinaryMoment Struct Reference . . . . .	39
5.7.1	Detailed Description . . . . .	40
5.8	BmPbil Class Reference . . . . .	40
5.8.1	Detailed Description . . . . .	42
5.8.2	Member Enumeration Documentation . . . . .	42
5.8.2.1	anonymous enum . . . . .	42
5.8.2.2	anonymous enum . . . . .	43
5.8.2.3	anonymous enum . . . . .	43
5.8.3	Member Function Documentation . . . . .	43
5.8.3.1	set_selection_size() . . . . .	43
5.9	Cache Class Reference . . . . .	44
5.9.1	Detailed Description . . . . .	45

5.9.2	Constructor & Destructor Documentation . . . . .	45
5.9.2.1	Cache() . . . . .	45
5.9.3	Member Function Documentation . . . . .	45
5.9.3.1	provides_incremental_evaluation() . . . . .	46
5.10	CallCounter Class Reference . . . . .	46
5.10.1	Detailed Description . . . . .	47
5.11	CompactGa Class Reference . . . . .	47
5.11.1	Detailed Description . . . . .	49
5.12	CompleteSearch Class Reference . . . . .	49
5.12.1	Detailed Description . . . . .	50
5.13	Crossover Class Reference . . . . .	50
5.13.1	Detailed Description . . . . .	50
5.13.2	Member Function Documentation . . . . .	51
5.13.2.1	breed() . . . . .	51
5.14	DeceptiveJump Class Reference . . . . .	51
5.14.1	Detailed Description . . . . .	52
5.14.2	Member Function Documentation . . . . .	52
5.14.2.1	get_maximum() . . . . .	52
5.14.2.2	has_known_maximum() . . . . .	53
5.15	EqualProducts Class Reference . . . . .	53
5.15.1	Detailed Description . . . . .	54
5.15.2	Member Function Documentation . . . . .	54
5.15.2.1	random() . . . . .	54
5.16	Error Class Reference . . . . .	55
5.16.1	Detailed Description . . . . .	56
5.17	ProgressTracker::Event Struct Reference . . . . .	56
5.17.1	Detailed Description . . . . .	56
5.18	Exception Class Reference . . . . .	56
5.18.1	Detailed Description . . . . .	57
5.19	Factorization Class Reference . . . . .	57

5.19.1 Detailed Description . . . . .	58
5.19.2 Constructor & Destructor Documentation . . . . .	58
5.19.2.1 Factorization() . . . . .	58
5.20 FirstAscentHillClimbing Class Reference . . . . .	59
5.20.1 Detailed Description . . . . .	60
5.21 FourPeaks Class Reference . . . . .	60
5.21.1 Detailed Description . . . . .	61
5.21.2 Member Function Documentation . . . . .	62
5.21.2.1 get_maximum() . . . . .	62
5.21.2.2 has_known_maximum() . . . . .	62
5.22 Function Class Reference . . . . .	63
5.22.1 Detailed Description . . . . .	64
5.22.2 Member Function Documentation . . . . .	64
5.22.2.1 compute_walsh_transform() . . . . .	64
5.22.2.2 get_maximum() . . . . .	65
5.22.2.3 incremental_eval() . . . . .	65
5.22.2.4 provides_incremental_evaluation() . . . . .	66
5.22.2.5 safe_eval() . . . . .	66
5.23 FunctionController Class Reference . . . . .	67
5.23.1 Detailed Description . . . . .	68
5.23.2 Member Function Documentation . . . . .	68
5.23.2.1 provides_incremental_evaluation() . . . . .	68
5.24 FunctionDecorator Class Reference . . . . .	68
5.24.1 Detailed Description . . . . .	69
5.25 FunctionMapComposition Class Reference . . . . .	69
5.25.1 Detailed Description . . . . .	70
5.25.2 Constructor & Destructor Documentation . . . . .	70
5.25.2.1 FunctionMapComposition() . . . . .	70
5.25.3 Member Function Documentation . . . . .	71
5.25.3.1 get_maximum() . . . . .	71

5.25.3.2	has_known_maximum()	71
5.26	FunctionModifier Class Reference	72
5.26.1	Detailed Description	72
5.27	FunctionPlugin Class Reference	73
5.27.1	Detailed Description	74
5.27.2	Constructor & Destructor Documentation	74
5.27.2.1	FunctionPlugin()	74
5.28	GeneticAlgorithm Class Reference	74
5.28.1	Detailed Description	76
5.28.2	Constructor & Destructor Documentation	76
5.28.2.1	GeneticAlgorithm()	76
5.28.3	Member Function Documentation	77
5.28.3.1	set_allow_stay()	77
5.29	HammingBall Class Reference	77
5.29.1	Detailed Description	78
5.29.2	Constructor & Destructor Documentation	78
5.29.2.1	HammingBall()	78
5.30	HammingSphere Class Reference	79
5.30.1	Detailed Description	80
5.30.2	Constructor & Destructor Documentation	80
5.30.2.1	HammingSphere()	80
5.31	HammingSphereIterator Class Reference	80
5.31.1	Detailed Description	81
5.31.2	Constructor & Destructor Documentation	82
5.31.2.1	HammingSphereIterator()	82
5.32	Hea< Moment, Herding > Class Template Reference	82
5.32.1	Detailed Description	84
5.32.2	Member Enumeration Documentation	85
5.32.2.1	anonymous enum	85
5.32.2.2	anonymous enum	85



5.32.3 Constructor & Destructor Documentation . . . . .	85
5.32.3.1 Hea() . . . . .	85
5.32.4 Member Function Documentation . . . . .	86
5.32.4.1 set_reset_period() . . . . .	86
5.32.4.2 set_selection_size() . . . . .	86
5.33 Hiff Class Reference . . . . .	87
5.33.1 Detailed Description . . . . .	87
5.33.2 Member Function Documentation . . . . .	88
5.33.2.1 get_maximum() . . . . .	88
5.33.2.2 has_known_maximum() . . . . .	88
5.34 Hypercubeliterator Class Reference . . . . .	88
5.34.1 Detailed Description . . . . .	89
5.35 IterativeAlgorithm Class Reference . . . . .	89
5.35.1 Detailed Description . . . . .	91
5.35.2 Constructor & Destructor Documentation . . . . .	91
5.35.2.1 IterativeAlgorithm() . . . . .	91
5.35.3 Member Function Documentation . . . . .	91
5.35.3.1 maximize() . . . . .	91
5.35.3.2 set_num_iterations() . . . . .	92
5.36 Iterator Class Reference . . . . .	92
5.36.1 Detailed Description . . . . .	93
5.37 Jump Class Reference . . . . .	94
5.37.1 Detailed Description . . . . .	94
5.37.2 Member Function Documentation . . . . .	95
5.37.2.1 get_maximum() . . . . .	95
5.37.2.2 has_known_maximum() . . . . .	95
5.38 Labs Class Reference . . . . .	95
5.38.1 Detailed Description . . . . .	96
5.39 LastEvaluation Class Reference . . . . .	96
5.39.1 Detailed Description . . . . .	97

5.40	LeadingOnes Class Reference	97
5.40.1	Detailed Description	98
5.40.2	Member Function Documentation	98
5.40.2.1	get_maximum()	98
5.40.2.2	has_known_maximum()	98
5.41	LinearFunction Class Reference	99
5.41.1	Detailed Description	100
5.41.2	Member Function Documentation	100
5.41.2.1	has_known_maximum()	100
5.41.2.2	random()	100
5.42	LinearMap Class Reference	101
5.42.1	Detailed Description	102
5.42.2	Member Function Documentation	102
5.42.2.1	is_surjective()	102
5.42.2.2	random()	102
5.43	LocalMaximum Class Reference	103
5.43.1	Detailed Description	103
5.44	LongPath Class Reference	104
5.44.1	Detailed Description	104
5.45	Map Class Reference	105
5.45.1	Detailed Description	106
5.45.2	Member Function Documentation	106
5.45.2.1	is_surjective()	106
5.46	MapComposition Class Reference	106
5.46.1	Detailed Description	107
5.46.2	Constructor & Destructor Documentation	107
5.46.2.1	MapComposition()	107
5.46.3	Member Function Documentation	108
5.46.3.1	is_surjective()	108
5.47	MaximumReached Class Reference	108

5.47.1 Detailed Description . . . . .	109
5.48 MaxSat Class Reference . . . . .	109
5.48.1 Detailed Description . . . . .	110
5.48.2 Member Function Documentation . . . . .	110
5.48.2.1 load() . . . . .	110
5.48.2.2 random() [1/2] . . . . .	110
5.48.2.3 random() [2/2] . . . . .	111
5.48.3 Member Data Documentation . . . . .	111
5.48.3.1 _expression . . . . .	111
5.49 Mmas Class Reference . . . . .	112
5.49.1 Detailed Description . . . . .	113
5.50 Model Class Reference . . . . .	113
5.50.1 Detailed Description . . . . .	114
5.51 ModelParameters Class Reference . . . . .	114
5.51.1 Detailed Description . . . . .	115
5.52 MuCommaLambdaEa Class Reference . . . . .	115
5.52.1 Detailed Description . . . . .	116
5.52.2 Constructor & Destructor Documentation . . . . .	116
5.52.2.1 MuCommaLambdaEa() . . . . .	116
5.52.3 Member Function Documentation . . . . .	117
5.52.3.1 set_allow_stay() . . . . .	117
5.53 MultiBitFlip Class Reference . . . . .	117
5.53.1 Detailed Description . . . . .	118
5.53.2 Constructor & Destructor Documentation . . . . .	118
5.53.2.1 MultiBitFlip() . . . . .	118
5.53.3 Member Function Documentation . . . . .	118
5.53.3.1 bernoulli_trials() . . . . .	118
5.53.3.2 reservoir_sampling() . . . . .	119
5.54 MuPlusLambdaEa Class Reference . . . . .	119
5.54.1 Detailed Description . . . . .	120

5.54.2	Constructor & Destructor Documentation	121
5.54.2.1	MuPlusLambdaEa()	121
5.54.3	Member Function Documentation	121
5.54.3.1	set_allow_stay()	121
5.55	Needle Class Reference	122
5.55.1	Detailed Description	122
5.55.2	Member Function Documentation	123
5.55.2.1	get_maximum()	123
5.55.2.2	has_known_maximum()	123
5.56	Negation Class Reference	124
5.56.1	Detailed Description	125
5.56.2	Member Function Documentation	125
5.56.2.1	get_maximum()	125
5.56.2.2	has_known_maximum()	125
5.56.2.3	provides_incremental_evaluation()	126
5.57	Neighborhood Class Reference	126
5.57.1	Detailed Description	128
5.57.2	Constructor & Destructor Documentation	128
5.57.2.1	Neighborhood()	128
5.57.3	Member Function Documentation	128
5.57.3.1	map()	128
5.57.3.2	mutate()	129
5.58	NeighborhoodIterator Class Reference	129
5.58.1	Detailed Description	130
5.58.2	Constructor & Destructor Documentation	130
5.58.2.1	NeighborhoodIterator()	130
5.59	NkLandscape Class Reference	130
5.59.1	Detailed Description	131
5.59.2	Member Function Documentation	132
5.59.2.1	random()	132

5.60 NpsPbil Class Reference . . . . .	132
5.60.1 Detailed Description . . . . .	134
5.61 OnBudgetFunction Class Reference . . . . .	134
5.61.1 Detailed Description . . . . .	136
5.61.2 Member Function Documentation . . . . .	136
5.61.2.1 eval() . . . . .	136
5.61.2.2 incremental_eval() . . . . .	136
5.61.2.3 update() . . . . .	137
5.62 OneMax Class Reference . . . . .	137
5.62.1 Detailed Description . . . . .	138
5.62.2 Member Function Documentation . . . . .	138
5.62.2.1 get_maximum() . . . . .	138
5.62.2.2 has_known_maximum() . . . . .	139
5.62.2.3 provides_incremental_evaluation() . . . . .	139
5.63 OnePlusLambdaCommaLambdaGa Class Reference . . . . .	139
5.63.1 Detailed Description . . . . .	140
5.63.2 Constructor & Destructor Documentation . . . . .	141
5.63.2.1 OnePlusLambdaCommaLambdaGa() . . . . .	141
5.64 OnePlusOneEa Class Reference . . . . .	141
5.64.1 Detailed Description . . . . .	142
5.64.2 Constructor & Destructor Documentation . . . . .	143
5.64.2.1 OnePlusOneEa() . . . . .	143
5.64.3 Member Function Documentation . . . . .	143
5.64.3.1 set_allow_stay() . . . . .	143
5.64.3.2 set_num_iterations() . . . . .	143
5.65 Pbil Class Reference . . . . .	144
5.65.1 Detailed Description . . . . .	145
5.66 Permutation Class Reference . . . . .	145
5.66.1 Detailed Description . . . . .	146
5.66.2 Member Function Documentation . . . . .	146

5.66.2.1	<a href="#">is_surjective()</a>	147
5.67	<a href="#">Plateau Class Reference</a>	147
5.67.1	<a href="#">Detailed Description</a>	148
5.67.2	<a href="#">Member Function Documentation</a>	148
5.67.2.1	<a href="#">get_maximum()</a>	148
5.67.2.2	<a href="#">has_known_maximum()</a>	148
5.68	<a href="#">PointValueException Class Reference</a>	149
5.68.1	<a href="#">Detailed Description</a>	149
5.69	<a href="#">Population Class Reference</a>	150
5.69.1	<a href="#">Detailed Description</a>	151
5.69.2	<a href="#">Member Function Documentation</a>	151
5.69.2.1	<a href="#">comma_selection()</a>	151
5.69.2.2	<a href="#">get_best_bv()</a> [1/2]	152
5.69.2.3	<a href="#">get_best_bv()</a> [2/2]	152
5.69.2.4	<a href="#">get_best_value()</a> [1/2]	152
5.69.2.5	<a href="#">get_best_value()</a> [2/2]	153
5.69.2.6	<a href="#">get_worst_bv()</a>	153
5.69.2.7	<a href="#">plus_selection()</a>	153
5.69.3	<a href="#">Member Data Documentation</a>	154
5.69.3.1	<a href="#">_compare_index_value</a>	154
5.69.3.2	<a href="#">_lookup</a>	154
5.70	<a href="#">PriorNoise Class Reference</a>	155
5.70.1	<a href="#">Detailed Description</a>	156
5.70.2	<a href="#">Member Function Documentation</a>	156
5.70.2.1	<a href="#">get_maximum()</a>	156
5.70.2.2	<a href="#">has_known_maximum()</a>	156
5.70.2.3	<a href="#">provides_incremental_evaluation()</a>	157
5.71	<a href="#">ProgressTracker Class Reference</a>	157
5.71.1	<a href="#">Detailed Description</a>	158
5.71.2	<a href="#">Member Function Documentation</a>	159

5.71.2.1	<a href="#">eval()</a>	159
5.71.2.2	<a href="#">get_last_improvement()</a>	159
5.71.2.3	<a href="#">incremental_eval()</a>	159
5.71.2.4	<a href="#">update()</a>	160
5.72	<a href="#">PvAlgorithm Class Reference</a>	160
5.72.1	<a href="#">Detailed Description</a>	161
5.72.2	<a href="#">Member Enumeration Documentation</a>	161
5.72.2.1	<a href="#">anonymous enum</a>	161
5.73	<a href="#">Qubo Class Reference</a>	162
5.73.1	<a href="#">Detailed Description</a>	163
5.73.2	<a href="#">Member Function Documentation</a>	163
5.73.2.1	<a href="#">load()</a>	163
5.73.3	<a href="#">Member Data Documentation</a>	164
5.73.3.1	<a href="#">_q</a>	164
5.74	<a href="#">Random Struct Reference</a>	164
5.74.1	<a href="#">Detailed Description</a>	164
5.75	<a href="#">RandomLocalSearch Class Reference</a>	165
5.75.1	<a href="#">Detailed Description</a>	166
5.75.2	<a href="#">Member Function Documentation</a>	166
5.75.2.1	<a href="#">set_patience()</a>	166
5.76	<a href="#">RandomSearch Class Reference</a>	167
5.76.1	<a href="#">Detailed Description</a>	168
5.77	<a href="#">Restart Class Reference</a>	168
5.77.1	<a href="#">Detailed Description</a>	169
5.78	<a href="#">Ridge Class Reference</a>	169
5.78.1	<a href="#">Detailed Description</a>	170
5.78.2	<a href="#">Member Function Documentation</a>	170
5.78.2.1	<a href="#">get_maximum()</a>	170
5.78.2.2	<a href="#">has_known_maximum()</a>	171
5.79	<a href="#">SimulatedAnnealing Class Reference</a>	171

5.79.1 Detailed Description . . . . .	173
5.79.2 Member Function Documentation . . . . .	173
5.79.2.1 init_beta() . . . . .	173
5.80 SingleBitFlip Class Reference . . . . .	173
5.80.1 Detailed Description . . . . .	174
5.81 SingleBitFlipIterator Class Reference . . . . .	174
5.81.1 Detailed Description . . . . .	175
5.81.2 Constructor & Destructor Documentation . . . . .	175
5.81.2.1 SingleBitFlipIterator() . . . . .	175
5.82 SinusSummationCancellation Class Reference . . . . .	176
5.82.1 Detailed Description . . . . .	176
5.83 SixPeaks Class Reference . . . . .	177
5.83.1 Detailed Description . . . . .	178
5.83.2 Member Function Documentation . . . . .	178
5.83.2.1 get_maximum() . . . . .	178
5.83.2.2 has_known_maximum() . . . . .	179
5.84 SpinHerdng Class Reference . . . . .	179
5.84.1 Detailed Description . . . . .	180
5.84.2 Member Enumeration Documentation . . . . .	181
5.84.2.1 anonymous enum . . . . .	181
5.84.3 Constructor & Destructor Documentation . . . . .	181
5.84.3.1 SpinHerdng() . . . . .	181
5.84.4 Member Function Documentation . . . . .	181
5.84.4.1 q_variation() . . . . .	181
5.85 SpinMoment Struct Reference . . . . .	182
5.85.1 Detailed Description . . . . .	183
5.86 SteepestAscentHillClimbing Class Reference . . . . .	183
5.86.1 Detailed Description . . . . .	184
5.87 StopOnMaximum Class Reference . . . . .	184
5.87.1 Detailed Description . . . . .	185



5.87.2	Constructor & Destructor Documentation	185
5.87.2.1	StopOnMaximum()	185
5.87.3	Member Function Documentation	186
5.87.3.1	eval()	186
5.87.3.2	incremental_eval()	186
5.87.3.3	update()	186
5.88	StopOnTarget Class Reference	187
5.88.1	Detailed Description	188
5.88.2	Constructor & Destructor Documentation	188
5.88.2.1	StopOnTarget()	188
5.88.3	Member Function Documentation	188
5.88.3.1	eval()	188
5.88.3.2	incremental_eval()	189
5.88.3.3	update()	189
5.89	SummationCancellation Class Reference	189
5.89.1	Detailed Description	191
5.89.2	Constructor & Destructor Documentation	191
5.89.2.1	SummationCancellation()	191
5.89.3	Member Function Documentation	191
5.89.3.1	has_known_maximum()	191
5.90	TargetReached Class Reference	192
5.90.1	Detailed Description	192
5.91	TournamentSelection Class Reference	193
5.91.1	Detailed Description	193
5.91.2	Member Function Documentation	194
5.91.2.1	select()	194
5.92	Translation Class Reference	194
5.92.1	Detailed Description	195
5.92.2	Member Function Documentation	195
5.92.2.1	is_surjective()	196

5.93	Trap Class Reference	196
5.93.1	Detailed Description	197
5.93.2	Constructor & Destructor Documentation	197
5.93.2.1	Trap()	197
5.93.3	Member Function Documentation	197
5.93.3.1	get_maximum()	198
5.93.3.2	has_known_maximum()	198
5.94	Umda Class Reference	198
5.94.1	Detailed Description	200
5.95	UniformCrossover Class Reference	200
5.95.1	Detailed Description	200
5.95.2	Member Function Documentation	201
5.95.2.1	breed()	201
5.96	WalshExpansion Class Reference	201
5.96.1	Detailed Description	202
5.96.2	Member Function Documentation	202
5.96.2.1	random()	202
5.97	WalshExpansion1 Class Reference	203
5.97.1	Detailed Description	204
5.97.2	Member Function Documentation	204
5.97.2.1	random()	204
5.98	WalshExpansion2 Class Reference	205
5.98.1	Detailed Description	206
5.98.2	Member Function Documentation	206
5.98.2.1	random()	206
5.98.3	Member Data Documentation	206
5.98.3.1	_quadratic	206
5.99	Function::WalshTransformTerm Struct Reference	207
5.99.1	Detailed Description	207
5.99.2	Member Data Documentation	207
5.99.2.1	feature	207

# 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 . . . . .	11
<a href="#">hnco::algorithm</a>	Algorithms . . . . .	20
<a href="#">hnco::algorithm::bm_pbil</a>	Boltzmann machine PBIL . . . . .	22
<a href="#">hnco::algorithm::hea</a>	Herding evolutionary algorithm . . . . .	22
<a href="#">hnco::exception</a>	Exceptions . . . . .	23
<a href="#">hnco::function</a>	Functions to be maximized . . . . .	23
<a href="#">hnco::neighborhood</a>	Neighborhoods for local search . . . . .	25
<a href="#">hnco::random</a>	Pseudo random numbers . . . . .	26



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Algorithm . . . . .	31
CompleteSearch . . . . .	49
IterativeAlgorithm . . . . .	89
BmPbil . . . . .	40
FirstAscentHillClimbing . . . . .	59
GeneticAlgorithm . . . . .	74
Hea< Moment, Herding > . . . . .	82
MuCommaLambdaEa . . . . .	115
MuPlusLambdaEa . . . . .	119
OnePlusLambdaCommaLambdaGa . . . . .	139
PvAlgorithm . . . . .	160
CompactGa . . . . .	47
Mmas . . . . .	112
NpsPbil . . . . .	132
Pbil . . . . .	144
Umda . . . . .	198
RandomLocalSearch . . . . .	165
RandomSearch . . . . .	167
Restart . . . . .	168
SimulatedAnnealing . . . . .	171
SteepestAscentHillClimbing . . . . .	183
OnePlusOneEa . . . . .	141
BinaryHerding . . . . .	37
BinaryMoment . . . . .	39
Crossover . . . . .	50
BiasedCrossover . . . . .	36
UniformCrossover . . . . .	200
ProgressTracker::Event . . . . .	56
Exception . . . . .	56
Error . . . . .	55
LastEvaluation . . . . .	96
PointValueException . . . . .	149
LocalMaximum . . . . .	103
MaximumReached . . . . .	108

TargetReached . . . . .	192
Function . . . . .	63
DeceptiveJump . . . . .	51
EqualProducts . . . . .	53
Factorization . . . . .	57
FourPeaks . . . . .	60
FunctionDecorator . . . . .	68
FunctionController . . . . .	67
Cache . . . . .	44
CallCounter . . . . .	46
OnBudgetFunction . . . . .	134
ProgressTracker . . . . .	157
StopOnMaximum . . . . .	184
StopOnTarget . . . . .	187
FunctionModifier . . . . .	72
AdditiveGaussianNoise . . . . .	27
FunctionMapComposition . . . . .	69
Negation . . . . .	124
PriorNoise . . . . .	155
FunctionPlugin . . . . .	73
Hiff . . . . .	87
Jump . . . . .	94
Labs . . . . .	95
LeadingOnes . . . . .	97
LinearFunction . . . . .	99
LongPath . . . . .	104
MaxSat . . . . .	109
Needle . . . . .	122
NkLandscape . . . . .	130
OneMax . . . . .	137
Plateau . . . . .	147
Qubo . . . . .	162
Ridge . . . . .	169
SixPeaks . . . . .	177
SummationCancellation . . . . .	189
SinusSummationCancellation . . . . .	176
Trap . . . . .	196
WalshExpansion . . . . .	201
WalshExpansion1 . . . . .	203
WalshExpansion2 . . . . .	205
Iterator . . . . .	92
Hypercubeliterator . . . . .	88
NeighborhoodIterator . . . . .	129
HammingSphereIterator . . . . .	80
SingleBitFlipIterator . . . . .	174
Map . . . . .	105
AffineMap . . . . .	29
LinearMap . . . . .	101
MapComposition . . . . .	106
Permutation . . . . .	145
Translation . . . . .	194
Model . . . . .	113
ModelParameters . . . . .	114
Neighborhood . . . . .	126
MultiBitFlip . . . . .	117
BernoulliProcess . . . . .	33
HammingBall . . . . .	77

HammingSphere . . . . .	79
SingleBitFlip . . . . .	173
Population . . . . .	150
TournamentSelection . . . . .	193
Random . . . . .	164
SpinHerdin . . . . .	179
SpinMoment . . . . .	182
Function::WalshTransformTerm . . . . .	207





## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">AdditiveGaussianNoise</a>	
Additive Gaussian Noise . . . . .	27
<a href="#">AffineMap</a>	
Affine map . . . . .	29
<a href="#">Algorithm</a>	
Abstract search algorithm . . . . .	31
<a href="#">BernoulliProcess</a>	
Bernoulli process . . . . .	33
<a href="#">BiasedCrossover</a>	
Biased crossover . . . . .	36
<a href="#">BinaryHerding</a>	
Herding with binary variables . . . . .	37
<a href="#">BinaryMoment</a>	
Moment for binary variables . . . . .	39
<a href="#">BmPbil</a>	
Boltzmann machine PBIL . . . . .	40
<a href="#">Cache</a>	
Cache . . . . .	44
<a href="#">CallCounter</a>	
Call counter . . . . .	46
<a href="#">CompactGa</a>	
Compact genetic algorithm . . . . .	47
<a href="#">CompleteSearch</a>	
Complete search . . . . .	49
<a href="#">Crossover</a>	
Crossover . . . . .	50
<a href="#">DeceptiveJump</a>	
Deceptive jump . . . . .	51
<a href="#">EqualProducts</a>	
Equal products . . . . .	53
<a href="#">Error</a>	
Error . . . . .	55
<a href="#">ProgressTracker::Event</a>	
Event . . . . .	56
<a href="#">Exception</a>	
Basic exception . . . . .	56

Factorization	
Factorization	57
FirstAscentHillClimbing	
First ascent hill climbing	59
FourPeaks	
Four Peaks	60
Function	
Function	63
FunctionController	
Function controller	67
FunctionDecorator	
Function decorator	68
FunctionMapComposition	
Composition of a function and a map	69
FunctionModifier	
Function modifier	72
FunctionPlugin	
Function plugin	73
GeneticAlgorithm	
Genetic algorithm	74
HammingBall	
Hamming ball	77
HammingSphere	
Hamming sphere	79
HammingSphereIterator	
Hamming sphere neighborhood iterator	80
Hea< Moment, Herding >	
Herding evolutionary algorithm	82
Hiff	
Hierarchical if and only if	87
HypercubeIterator	
Hypercube iterator	88
IterativeAlgorithm	
Iterative search	89
Iterator	
Iterator over bit vectors	92
Jump	
Jump	94
Labs	
Low autocorrelation binary sequences	95
LastEvaluation	
Last evaluation	96
LeadingOnes	
Leading ones	97
LinearFunction	
Linear function	99
LinearMap	
Linear map	101
LocalMaximum	
Local maximum	103
LongPath	
Long path	104
Map	
Map	105
MapComposition	
Map composition	106
MaximumReached	
Maximum reached	108

MaxSat	
MAX-SAT	109
Mmas	
Max-min ant system	112
Model	
Model of a Boltzmann machine	113
ModelParameters	
Parameters of a Boltzmann machine	114
MuCommaLambdaEa	
(mu, lambda) EA	115
MultiBitFlip	
Multi bit flip	117
MuPlusLambdaEa	
(mu+lambda) EA	119
Needle	
Needle in a haystack	122
Negation	
Negation	124
Neighborhood	
Neighborhood	126
NeighborhoodIterator	
Neighborhood iterator	129
NkLandscape	
NK landscape	130
NpsPbil	
Population-based incremental learning with negative and positive selection	132
OnBudgetFunction	
CallCounter with a limited number of evaluations	134
OneMax	
OneMax	137
OnePlusLambdaCommaLambdaGa	
(1+(lambda, lambda)) genetic algorithm	139
OnePlusOneEa	
(1+1) EA	141
Pbil	
Population-based incremental learning	144
Permutation	
Permutation	145
Plateau	
Plateau	147
PointValueException	
Point-value exception	149
Population	
Population	150
PriorNoise	
Prior noise	155
ProgressTracker	
ProgressTracker	157
PvAlgorithm	
Probability vector algorithm	160
Qubo	
Quadratic unconstrained binary optimization	162
Random	
Random numbers	164
RandomLocalSearch	
Random local search	165
RandomSearch	
Random search	167

Restart	
Restart	168
Ridge	
Ridge	169
SimulatedAnnealing	
Simulated annealing	171
SingleBitFlip	
One bit neighborhood	173
SingleBitFlipIterator	
Single bit flip neighborhood iterator	174
SinusSummationCancellation	
Summation cancellation with sinus	176
SixPeaks	
Six Peaks	177
SpinHerdin	
Herdin with spin variables	179
SpinMoment	
Moment for spin variables	182
SteepestAscentHillClimbing	
Steepest ascent hill climbing	183
StopOnMaximum	
Stop on maximum	184
StopOnTarget	
Stop on target	187
SummationCancellation	
Summation cancellation	189
TargetReached	
Target reached	192
TournamentSelection	
Population with tournament selection	193
Translation	
Translation	194
Trap	
Trap	196
Umda	
Univariate marginal distribution algorithm	198
UniformCrossover	
Uniform crossover	200
WalshExpansion	
Walsh expansion	201
WalshExpansion1	
Walsh expansion of degree 1	203
WalshExpansion2	
Walsh expansion of degree 2	205
Function::WalshTransformTerm	
Walsh transform term	207

## 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 [Iterator](#)  
*Iterator over bit vectors.*
- class [LinearMap](#)  
*Linear map.*
- class [Map](#)  
*Map.*
- class [MapComposition](#)  
*Map composition.*
- class [Permutation](#)  
*Permutation.*
- class [Translation](#)  
*Translation.*

## 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.*

### 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\_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 HNCO 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 52 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 94 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 29 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

<i>M</i>	input matrix
<i>N</i>	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 200 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 242 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 133 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 103 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 150 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 181 of file bit-matrix.cc.

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

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

Convert a bool vector to a bit vector.

**Warning**

Vectors must be of the same size.

Definition at line 208 of file bit-vector.hh.

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

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

Convert a bit vector to a bool vector.

**Warning**

Vectors must be of the same size.

Definition at line 192 of file bit-vector.hh.

**4.1.3.11 sbm\_multiply()**

```
void hnco::sbm_multiply (
    const sparse_bit_matrix_t & M,
    const bit_vector_t & x,
    bit_vector_t & y ) [inline]
```

Multiply a sparse bit matrix and a bit vector.

The result is  $y = Mx$ .

Definition at line 68 of file sparse-bit-matrix.hh.

## 4.2 hnco::algorithm Namespace Reference

Algorithms.

### Namespaces

- [bm\\_pbil](#)  
*Boltzmann machine PBIL.*
- [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 [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 [PvAlgorithm](#)  
*Probability vector algorithm.*
- class [RandomLocalSearch](#)  
*Random local search.*

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

## 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::hea` Namespace Reference

Herding evolutionary algorithm.



## Classes

- class [BinaryHerding](#)  
*Herding with binary variables.*
- struct [BinaryMoment](#)  
*Moment for binary variables.*
- class [Hea](#)  
*Herding evolutionary algorithm.*
- class [SpinHerding](#)  
*Herding with spin variables.*
- struct [SpinMoment](#)  
*Moment for spin variables.*

### 4.4.1 Detailed Description

Herding evolutionary algorithm.

## 4.5 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*

### 4.5.1 Detailed Description

Exceptions.

## 4.6 hnco::function Namespace Reference

Functions to be maximized.

## Classes

- 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 [LeadingOnes](#)  
*Leading ones.*
- class [LinearFunction](#)  
*Linear function.*
- class [LongPath](#)  
*Long path.*
- 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.*

### 4.6.1 Detailed Description

Functions to be maximized.

## 4.7 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.*

### 4.7.1 Detailed Description

Neighborhoods for local search.

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

## 4.8 `hnco::random` Namespace Reference

Pseudo random numbers.

## Classes

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

### 4.8.1 Detailed Description

Pseudo random numbers.

## Chapter 5

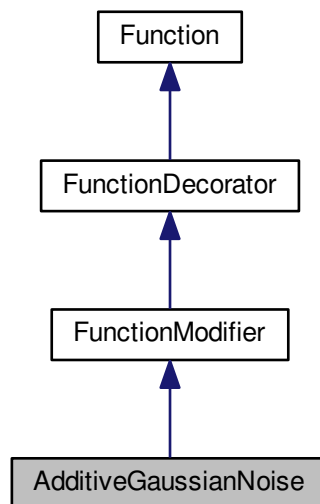
# Class Documentation

### 5.1 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

#### 5.1.1 Detailed Description

Additive Gaussian Noise.

Definition at line 166 of file function-modifier.hh.

#### 5.1.2 Member Function Documentation

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

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

Get the global maximum.

#### Exceptions

Error	
-------	--

Reimplemented from [Function](#).

Definition at line 188 of file function-modifier.hh.

##### 5.1.2.2 [has\\_known\\_maximum\(\)](#)

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

Check for a known maximum.

**Returns**

false

Reimplemented from [Function](#).

Definition at line 192 of file function-modifier.hh.

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

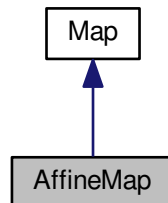
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

## 5.2 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**

### 5.2.1 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.

### 5.2.2 Member Function Documentation

#### 5.2.2.1 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 136 of file map.cc.

#### 5.2.2.2 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 99 of file map.cc.

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

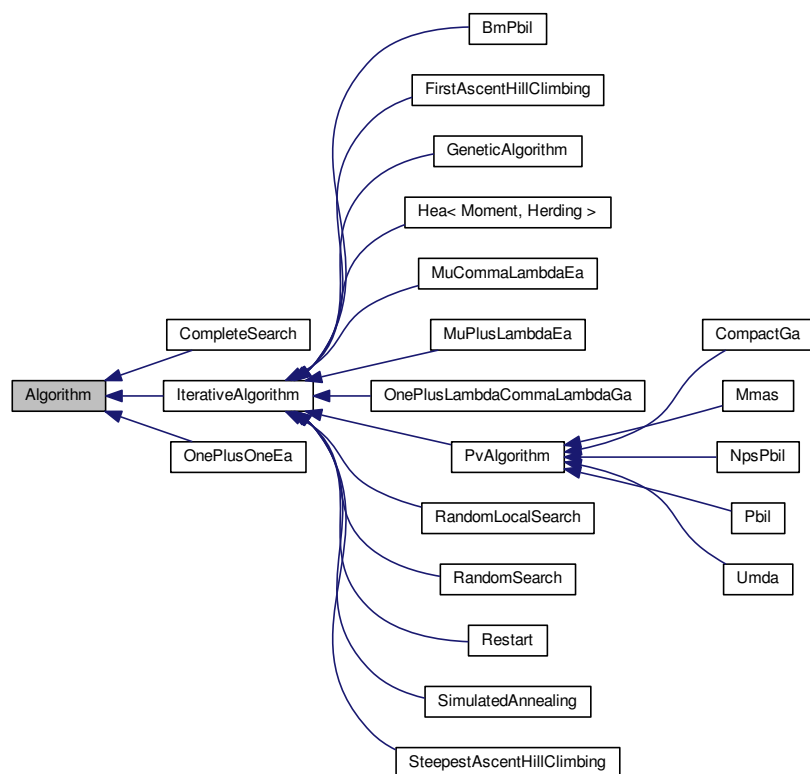
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.3 Algorithm Class Reference

Abstract search algorithm.

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

Inheritance diagram for Algorithm:



## Public Member Functions

- [Algorithm](#) ()  
*Constructor.*
- [Algorithm](#) (int n)  
*Constructor.*
- virtual [~Algorithm](#) ()  
*Destructor.*
- virtual void [init](#) ()  
*Initialization.*
- virtual void [maximize](#) ()=0  
*Maximize.*
- virtual const [point\\_value\\_t](#) & [get\\_solution](#) ()  
*Solution.*

## 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.*

## Protected Member Functions

- 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.*

## Parameters

- std::ostream \* [\\_stream](#) = &std::cout  
*Output stream.*

### 5.3.1 Detailed Description

Abstract search algorithm.

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

Definition at line 38 of file algorithm.hh.

### 5.3.2 Member Data Documentation

#### 5.3.2.1 \_functions

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

Functions.

Each thread has its own function.

Definition at line 49 of file algorithm.hh.

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

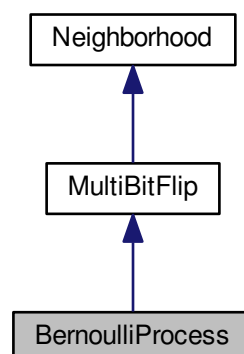
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

## 5.4 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

### 5.4.1 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.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 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.

## 5.4.2.2 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.

## 5.4.3 Member Function Documentation

## 5.4.3.1 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.

## 5.4.3.2 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:

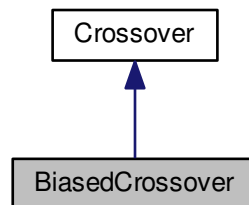
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.5 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.*

#### 5.5.1 Detailed Description

Biased crossover.

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

#### 5.5.2 Member Function Documentation

##### 5.5.2.1 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

## 5.6 BinaryHerding Class Reference

Herding with binary variables.

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

### Public Types

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

### Public Member Functions

- [BinaryHerding](#) (int n)  
*Constructor.*
- void [init](#) ()  
*Initialization.*
- void [sample](#) (const [BinaryMoment](#) &target, [bit\\_vector\\_t](#) &x)  
*Sample a bit vector.*
- double [error](#) (const [BinaryMoment](#) &target)  
*Compute the error.*
- double [delta](#) (const [BinaryMoment](#) &target)  
*Compute the norm of 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 `BinaryMoment` &target)  
*Compute delta.*
- void `sample_minimize_norm` (const `BinaryMoment` &target, `bit_vector_t` &x)  
*Sample a bit vector.*
- void `sample_maximize_inner_product` (const `BinaryMoment` &target, `bit_vector_t` &x)  
*Sample a bit vector.*

## Protected Attributes

- `BinaryMoment _count`  
*Counter moment.*
- `BinaryMoment _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.*

## 5.6.1 Detailed Description

Herding with binary variables.

Definition at line 38 of file herding-binary.hh.

## 5.6.2 Member Enumeration Documentation

### 5.6.2.1 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 herding-binary.hh.

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

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

## 5.7 BinaryMoment Struct Reference

Moment for binary variables.

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

### Public Member Functions

- [BinaryMoment](#) (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 [BinaryMoment](#) &p, double rate)  
*Update moment.*
- void [bound](#) (double margin)  
*Bound moment.*
- double [distance](#) (const [BinaryMoment](#) &p) const  
*Distance.*
- double [norm\\_2](#) () const  
*Compute the norm 2.*
- double [diameter](#) () const  
*Compute the diameter.*
- size\_t [size](#) () const  
*Size.*

### Public Attributes

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

### 5.7.1 Detailed Description

Moment for binary variables.

Definition at line 37 of file moment-binary.hh.

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

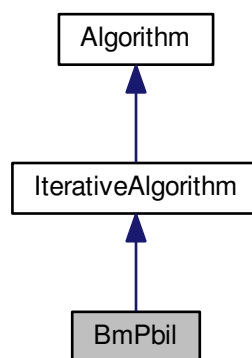
- lib/hnco/algorithms/hea/moment-binary.hh
- lib/hnco/algorithms/hea/moment-binary.cc

## 5.8 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\_SYNC\_NCHRONOUS** }
- 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 _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_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

### 5.8.1 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.

### 5.8.2 Member Enumeration Documentation

#### 5.8.2.1 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.

### 5.8.2.2 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.

### 5.8.2.3 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.

## 5.8.3 Member Function Documentation

### 5.8.3.1 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:

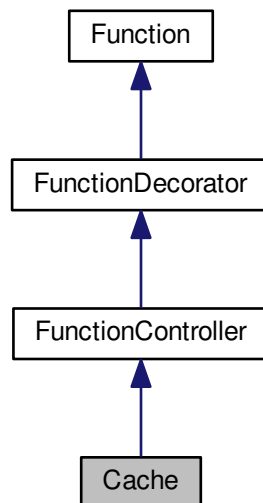
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

## 5.9 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.*

### 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

### 5.9.1 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 355 of file `function-controller.hh`.

### 5.9.2 Constructor & Destructor Documentation

#### 5.9.2.1 `Cache()`

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

Constructor.

Parameters

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

Definition at line 374 of file `function-controller.hh`.

### 5.9.3 Member Function Documentation

### 5.9.3.1 provides\_incremental\_evaluation()

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

Check whether the function provides incremental evaluation.

#### Returns

false

Reimplemented from [FunctionController](#).

Definition at line 383 of file function-controller.hh.

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

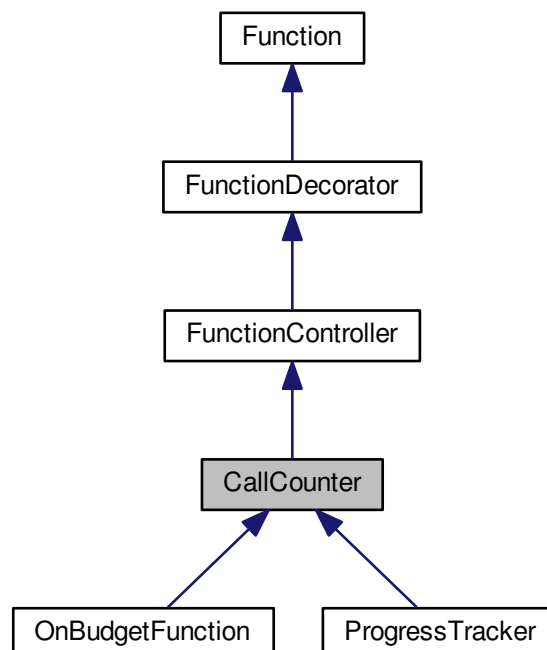
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.10 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.*

### 5.10.1 Detailed Description

Call counter.

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

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

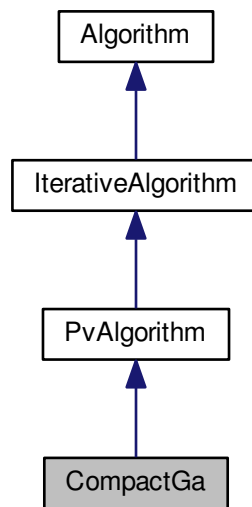
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

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

## Additional Inherited Members

### 5.11.1 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:

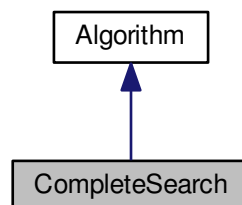
- lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

## 5.12 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

### 5.12.1 Detailed Description

Complete search.

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

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

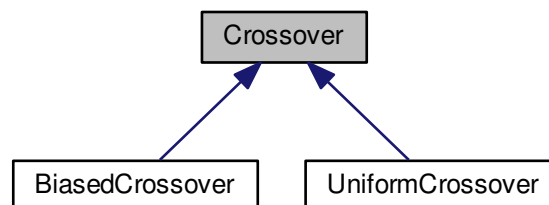
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

## 5.13 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.*

### 5.13.1 Detailed Description

[Crossover](#).

Definition at line 35 of file crossover.hh.

### 5.13.2 Member Function Documentation

#### 5.13.2.1 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:

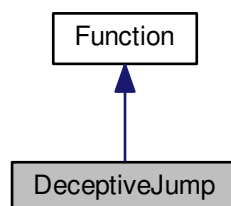
- `lib/hnco/algorithms/ea/crossover.hh`

## 5.14 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.*

### 5.14.1 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. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 84 of file jump.hh.

### 5.14.2 Member Function Documentation

#### 5.14.2.1 [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.

#### 5.14.2.2 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:

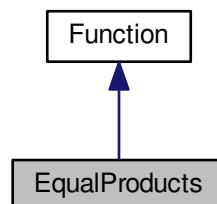
- lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

## 5.15 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`**

### 5.15.1 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.

### 5.15.2 Member Function Documentation

#### 5.15.2.1 `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:

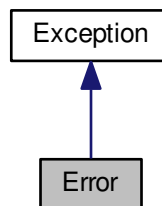
- lib/hnco/functions/equal-products.hh
- lib/hnco/functions/equal-products.cc

## 5.16 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.*

### 5.16.1 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

## 5.17 ProgressTracker::Event Struct Reference

[Event.](#)

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

### Public Attributes

- int [time](#)  
*Time.*
- double [value](#)  
*Value.*

### 5.17.1 Detailed Description

[Event.](#)

Definition at line 218 of file function-controller.hh.

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

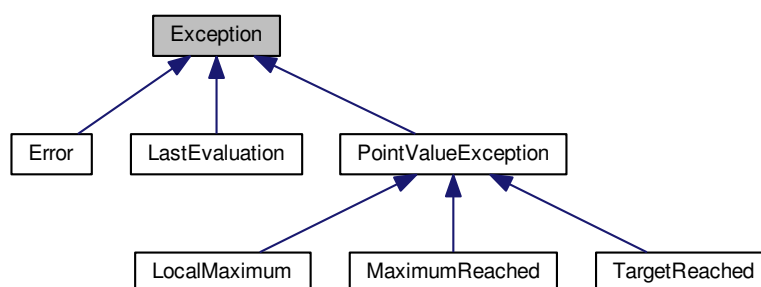
- lib/hnco/functions/decorators/function-controller.hh

## 5.18 Exception Class Reference

Basic exception.

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

Inheritance diagram for Exception:



### 5.18.1 Detailed Description

Basic exception.

Definition at line 35 of file exception.hh.

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

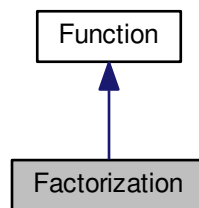
- lib/hnco/exception.hh

## 5.19 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.*

### 5.19.1 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.

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 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:

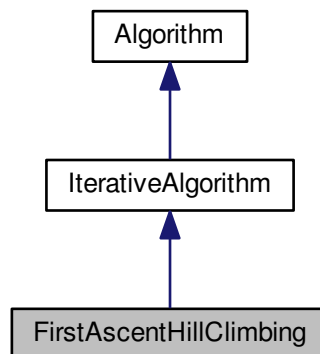
- lib/hnco/functions/factorization.hh
- lib/hnco/functions/factorization.cc

## 5.20 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.*

### 5.20.1 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:

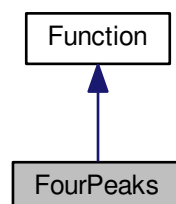
- `lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh`
- `lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc`

## 5.21 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.*

### 5.21.1 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.

## 5.21.2 Member Function Documentation

### 5.21.2.1 `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.

### 5.21.2.2 `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

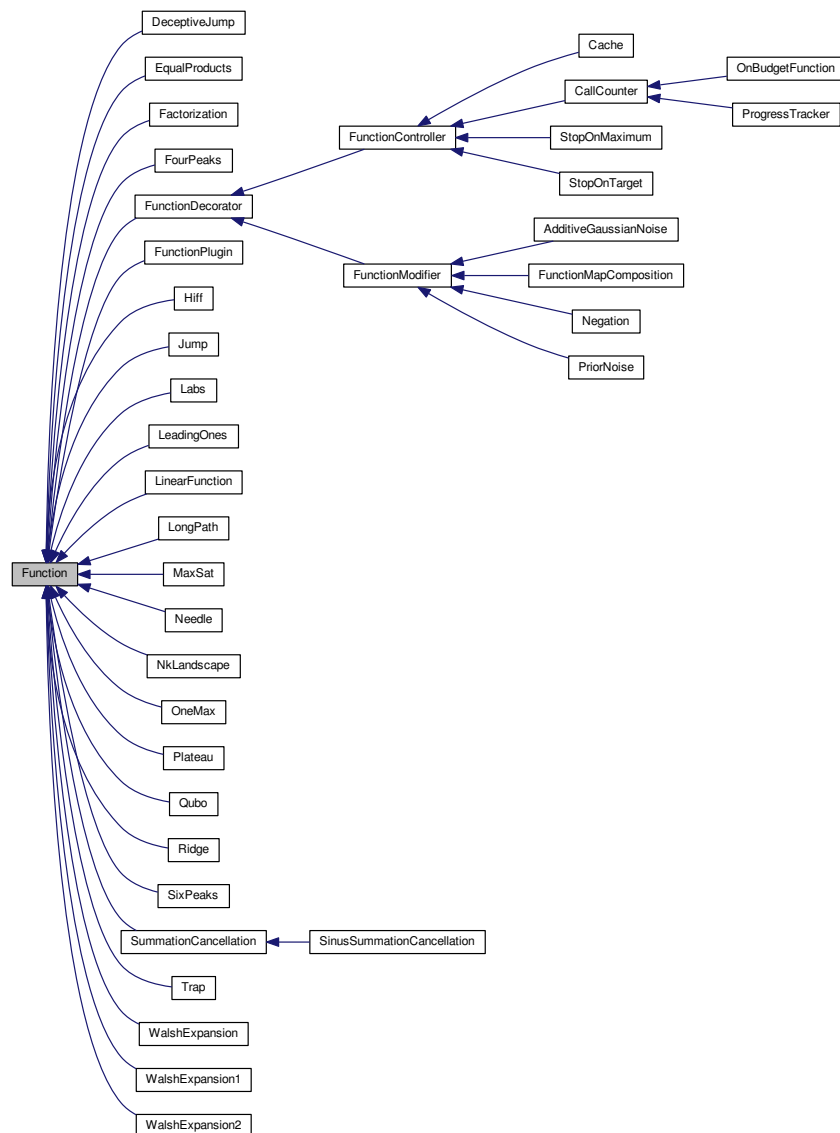


## 5.22 Function Class Reference

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.*

## 5.22.1 Detailed Description

[Function.](#)

Definition at line 39 of file `function.hh`.

## 5.22.2 Member Function Documentation

### 5.22.2.1 [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 33 of file function.cc.

5.22.2.2 `get_maximum()`

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

Get the global maximum.

## Exceptions

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

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

Definition at line 78 of file function.hh.

5.22.2.3 `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 131 of file function.hh.

#### 5.22.2.4 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 86 of file function.hh.

#### 5.22.2.5 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 141 of file function.hh.

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

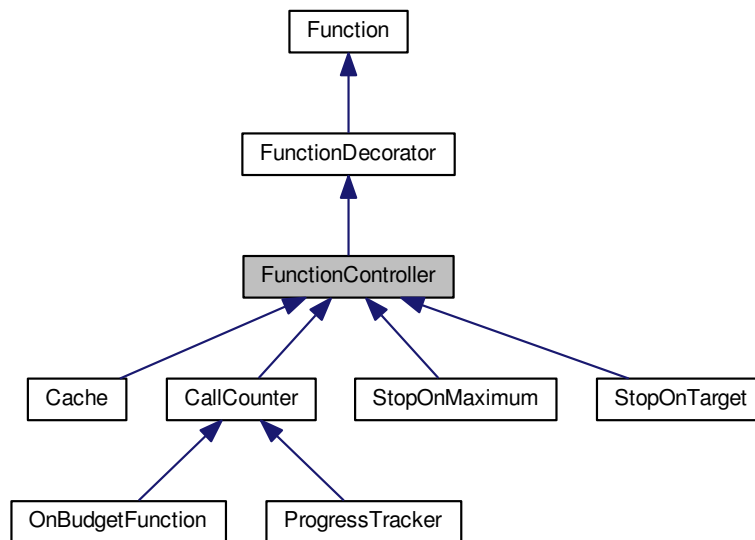
- [lib/hnco/functions/function.hh](#)
- [lib/hnco/functions/function.cc](#)

## 5.23 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.*

### Display

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

## Additional Inherited Members

### 5.23.1 Detailed Description

[Function](#) controller.

Definition at line 38 of file function-controller.hh.

### 5.23.2 Member Function Documentation

#### 5.23.2.1 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 63 of file function-controller.hh.

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

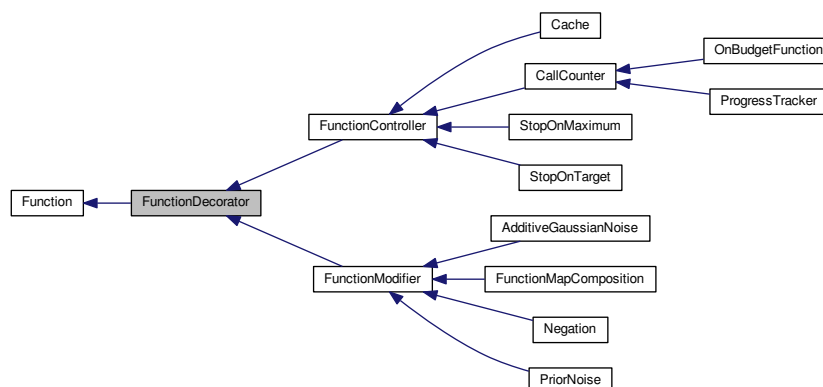
- lib/hnco/functions/decorators/function-controller.hh

## 5.24 FunctionDecorator Class Reference

[Function](#) decorator.

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

Inheritance diagram for FunctionDecorator:



### Public Member Functions

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

### Protected Attributes

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

#### 5.24.1 Detailed Description

[Function](#) decorator.

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

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

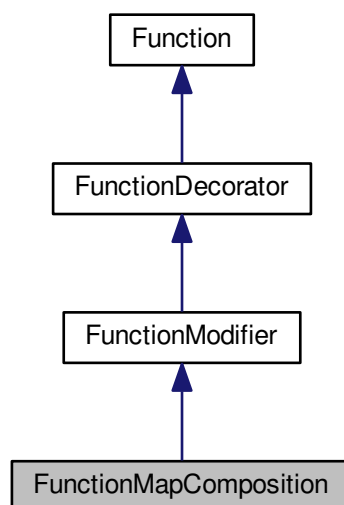
- lib/hnco/functions/decorators/function-decorator.hh

## 5.25 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.*

## Private Attributes

- [Map](#) \* [\\_map](#)  
*Map.*
- [bit\\_vector\\_t](#) [\\_bv](#)  
*\_bv*

## Additional Inherited Members

### 5.25.1 Detailed Description

Composition of a function and a map.

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

### 5.25.2 Constructor & Destructor Documentation

#### 5.25.2.1 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.

### 5.25.3 Member Function Documentation

#### 5.25.3.1 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.

#### 5.25.3.2 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:

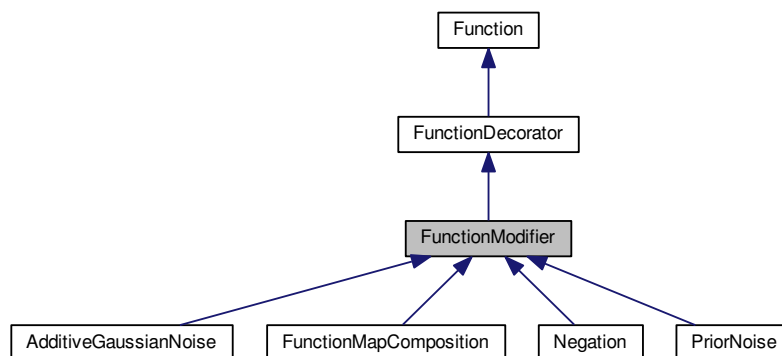
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

## 5.26 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

#### 5.26.1 Detailed Description

Function modifier.

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

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

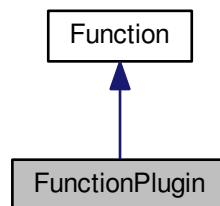
- `lib/hnco/functions/decorators/function-modifier.hh`

## 5.27 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.*

### 5.27.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

### 5.27.2 Constructor & Destructor Documentation

#### 5.27.2.1 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:

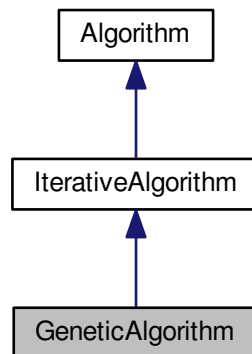
- lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

## 5.28 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

### 5.28.1 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.

### 5.28.2 Constructor & Destructor Documentation

#### 5.28.2.1 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.

### 5.28.3 Member Function Documentation

#### 5.28.3.1 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:

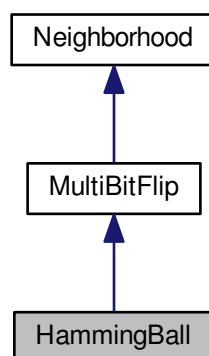
- lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

## 5.29 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

### 5.29.1 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.

### 5.29.2 Constructor & Destructor Documentation

#### 5.29.2.1 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:



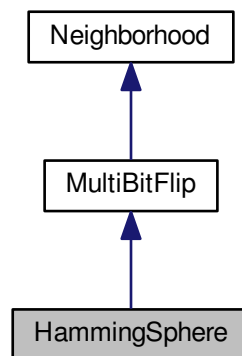
- `lib/hnco/neighborhoods/neighborhood.hh`
- `lib/hnco/neighborhoods/neighborhood.cc`

## 5.30 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

### 5.30.1 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.

### 5.30.2 Constructor & Destructor Documentation

#### 5.30.2.1 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:

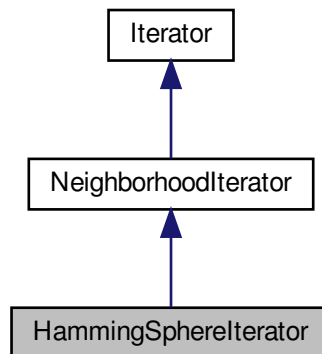
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.31 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

#### 5.31.1 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.

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 HammingSphereIterator()

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

Constructor.

#### Parameters

<i>n</i>	Size of bit vectors
<i>r</i>	Radius of Hamming Ball

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

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

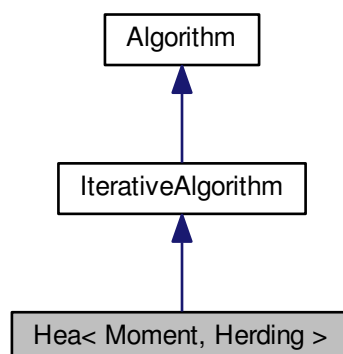
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

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

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



## Public Types

- enum { [RATE\\_CONSTANT](#), [RATE\\_EXPONENTIAL](#), [RATE\\_INVERSE](#) }
- enum { [LOG\\_ERROR](#), [LOG\\_DTU](#), [LOG\\_DELTA](#), [LOG\\_SELECTION](#), [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\\_rate\\_strategy](#) (int x)  
*Set the rate strategy.*
- void [set\\_reset\\_period](#) (int x)  
*Set the reset period.*
- void [set\\_delay](#) (int x)  
*Set the delay.*
- void [set\\_initial\\_rate](#) (double x)  
*Set the initial value of the learning rate.*
- void [set\\_time\\_constant](#) (double x)  
*Set the time constant.*
- 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.*
- 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 [\\_rate\\_strategy](#) = [RATE\\_CONSTANT](#)  
*Rate strategy.*
- int [\\_reset\\_period](#) = 0  
*Reset period.*
- int [\\_delay](#) = 10000  
*Delay.*
- double [\\_initial\\_rate](#) = 1e-4  
*Initial value of the learning rate.*
- double [\\_time\\_constant](#) = 1000  
*Time constant.*
- bool [\\_bound\\_moment](#) = false  
*Bound moment after update.*

## Additional Inherited Members

### 5.32.1 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 49 of file hea.hh.

### 5.32.2 Member Enumeration Documentation

#### 5.32.2.1 anonymous enum

anonymous enum

##### Enumerator

RATE_CONSTANT	Constant rate.
RATE_EXPONENTIAL	Exponential decay.
RATE_INVERSE	Inverse decay.

Definition at line 54 of file hea.hh.

#### 5.32.2.2 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.

Definition at line 65 of file hea.hh.

### 5.32.3 Constructor & Destructor Documentation

#### 5.32.3.1 Hea()

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

Constructor.

##### Parameters

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

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

Definition at line 234 of file `hea.hh`.

## 5.32.4 Member Function Documentation

### 5.32.4.1 `set_reset_period()`

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

Set the reset period.

#### Parameters

<code>x</code>	Reset period
----------------	--------------

$x \leq 0$  means no reset.

Definition at line 281 of file `hea.hh`.

### 5.32.4.2 `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 270 of file `hea.hh`.

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

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

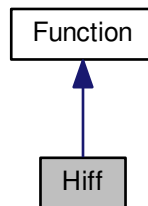


## 5.33 Hiff Class Reference

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.*

#### 5.33.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 165 of file theory.hh.

### 5.33.2 Member Function Documentation

#### 5.33.2.1 `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 191 of file theory.hh.

#### 5.33.2.2 `has_known_maximum()`

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

Check for a known maximum.

##### Returns

true

Reimplemented from [Function](#).

Definition at line 187 of file theory.hh.

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

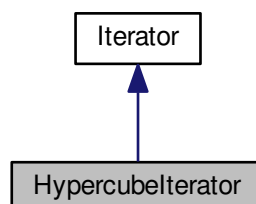
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.34 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

### 5.34.1 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:

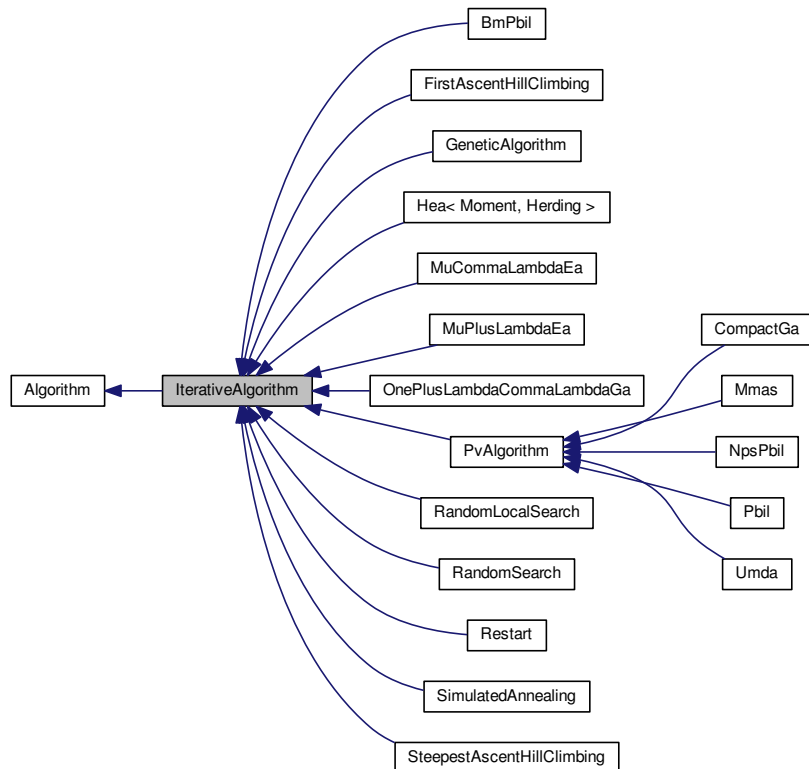
- lib/hnco/iterator.hh
- lib/hnco/iterator.cc

## 5.35 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.*

### 5.35.1 Detailed Description

Iterative search.

Definition at line 130 of file `algorithm.hh`.

### 5.35.2 Constructor & Destructor Documentation

#### 5.35.2.1 IterativeAlgorithm()

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

Constructor.

#### Parameters

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

Definition at line 160 of file `algorithm.hh`.

### 5.35.3 Member Function Documentation

#### 5.35.3.1 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.

#### 5.35.3.2 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 184 of file algorithm.hh.

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

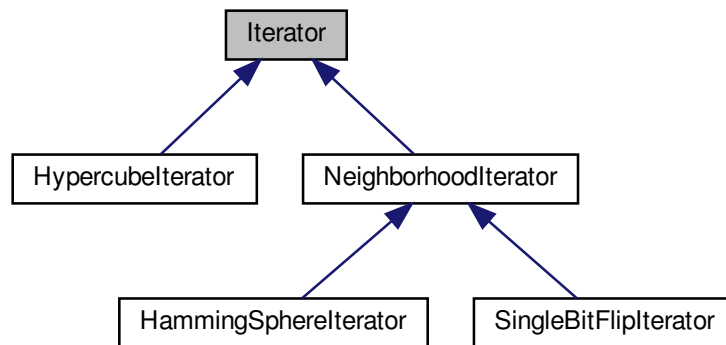
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

## 5.36 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.*

### 5.36.1 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:

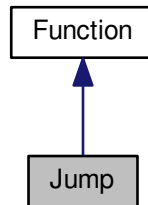
- lib/hnco/iterator.hh

## 5.37 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.*

### 5.37.1 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.



## 5.37.2 Member Function Documentation

### 5.37.2.1 `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`.

### 5.37.2.2 `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:

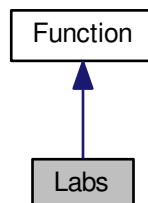
- `lib/hnco/functions/jump.hh`
- `lib/hnco/functions/jump.cc`

## 5.38 Labs Class Reference

Low autocorrelation binary sequences.

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

Inheritance diagram for Labs:



## Public Member Functions

- [Labs](#) (int n)  
*Constructor.*
- `size_t` [get\\_bv\\_size](#) ()  
*Get bit vector size.*
- `double` [eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

## Private Attributes

- `std::vector< int >` [\\_sequence](#)  
*Binary sequence written using 1 and -1.*

### 5.38.1 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 43 of file labs.hh.

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

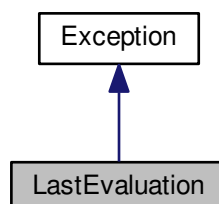
- lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

## 5.39 LastEvaluation Class Reference

Last evaluation.

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

Inheritance diagram for LastEvaluation:



### 5.39.1 Detailed Description

Last evaluation.

Definition at line 79 of file exception.hh.

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

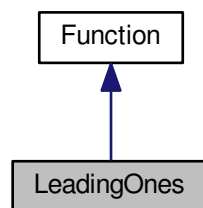
- lib/hnco/exception.hh

## 5.40 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.*

### 5.40.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 93 of file theory.hh.

### 5.40.2 Member Function Documentation

#### 5.40.2.1 `get_maximum()`

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

Get the global maximum.

Returns

`_bv_size`

Reimplemented from [Function](#).

Definition at line 117 of file theory.hh.

#### 5.40.2.2 `has_known_maximum()`

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

Check for a known maximum.

Returns

`true`

Reimplemented from [Function](#).

Definition at line 113 of file theory.hh.

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

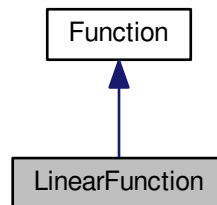
- `lib/hnco/functions/theory.hh`
- `lib/hnco/functions/theory.cc`

## 5.41 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**

### 5.41.1 Detailed Description

Linear function.

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

### 5.41.2 Member Function Documentation

#### 5.41.2.1 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.

#### 5.41.2.2 random()

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

Random instance.

#### Parameters

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

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

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

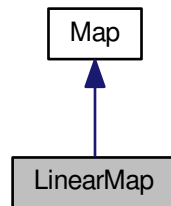
- lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

## 5.42 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**

### 5.42.1 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.

### 5.42.2 Member Function Documentation

#### 5.42.2.1 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 90 of file map.cc.

#### 5.42.2.2 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 61 of file map.cc.

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

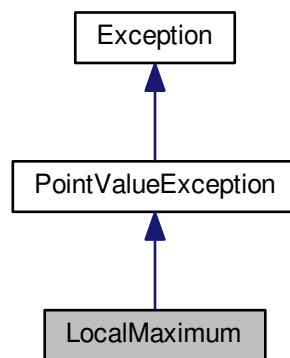
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.43 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

#### 5.43.1 Detailed Description

Local maximum.

Definition at line 70 of file exception.hh.

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

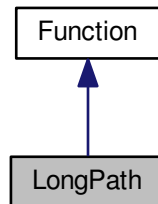
- lib/hnco/exception.hh

## 5.44 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.*
- [size\\_t get\\_bv\\_size](#) ()  
*Get bit vector size.*
- [double eval](#) (const [bit\\_vector\\_t](#) &)  
*Evaluate a bit vector.*

### Private Attributes

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

#### 5.44.1 Detailed Description

Long path.

Long paths have been introduced by Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Here we 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.

References:

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

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 58 of file long-path.hh.

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

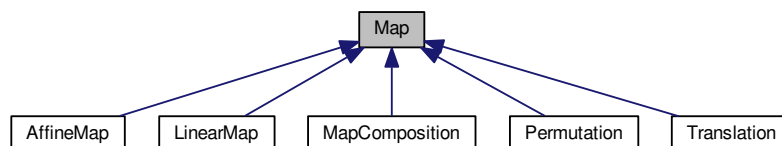
- lib/hnco/functions/long-path.hh
- lib/hnco/functions/long-path.cc

## 5.45 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.*

### 5.45.1 Detailed Description

[Map](#).

Definition at line 39 of file map.hh.

### 5.45.2 Member Function Documentation

#### 5.45.2.1 `is_surjective()`

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

Check for surjective map.

#### Returns

false

Reimplemented in [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:

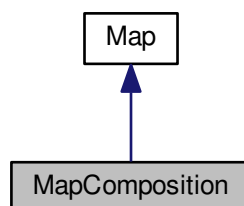
- lib/hnco/map.hh

## 5.46 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.*

### 5.46.1 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.

### 5.46.2 Constructor & Destructor Documentation

#### 5.46.2.1 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.

### 5.46.3 Member Function Documentation

#### 5.46.3.1 [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:

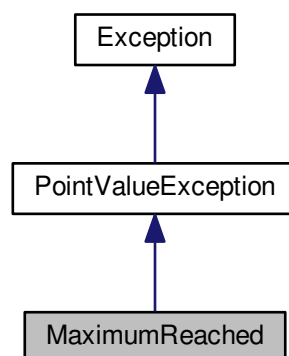
- lib/hnco/map.hh

### 5.47 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

### 5.47.1 Detailed Description

Maximum reached.

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

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

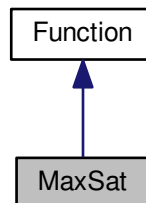
- `lib/hnco/exception.hh`

## 5.48 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.*
- void [load](#) (std::istream &stream)  
*Load an instance.*
- void [save](#) (std::ostream &stream)  
*Save an instance.*
- 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 the expression.*

## Private Attributes

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

### 5.48.1 Detailed Description

MAX-SAT.

Reference:

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

Definition at line 42 of file max-sat.hh.

### 5.48.2 Member Function Documentation

#### 5.48.2.1 load()

```
void load (
    std::istream & stream )
```

Load an instance.

#### Exceptions

Error	
-------	--

Definition at line 133 of file max-sat.cc.

#### 5.48.2.2 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 38 of file max-sat.cc.

### 5.48.2.3 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 66 of file max-sat.cc.

## 5.48.3 Member Data Documentation

### 5.48.3.1 \_expression

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

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 52 of file max-sat.hh.

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

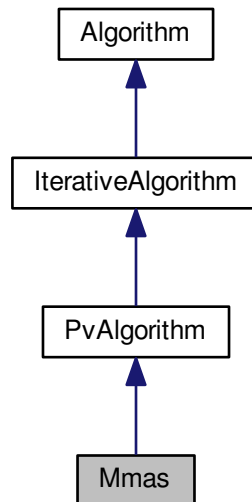
- lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

## 5.49 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\\_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 _rate = 1e-3`  
*Learning rate.*

## Additional Inherited Members

### 5.49.1 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`

## 5.50 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.*

## 5.50.1 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`

## 5.51 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**

### 5.51.1 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:

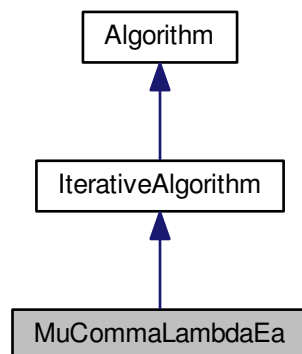
- lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

## 5.52 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

### 5.52.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

### 5.52.2 Constructor & Destructor Documentation

#### 5.52.2.1 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.

### 5.52.3 Member Function Documentation

#### 5.52.3.1 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:

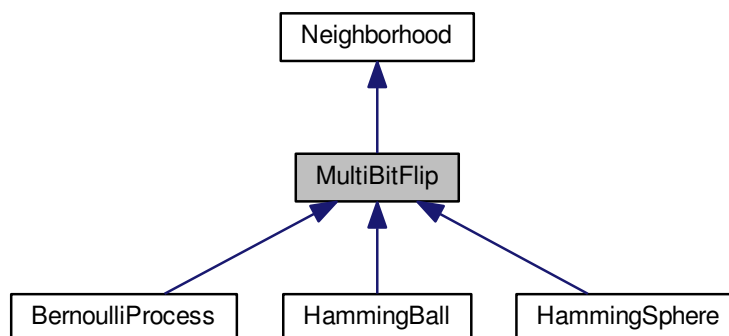
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

## 5.53 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

### 5.53.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

### 5.53.2 Constructor & Destructor Documentation

#### 5.53.2.1 MultiBitFlip()

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

Constructor.

#### Parameters

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

Definition at line 206 of file neighborhood.hh.

### 5.53.3 Member Function Documentation

#### 5.53.3.1 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.

**5.53.3.2 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:

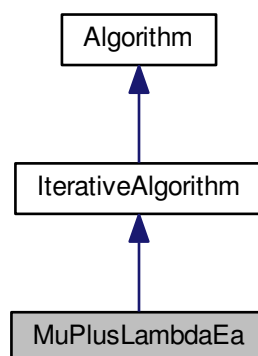
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.54 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

### 5.54.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

## 5.54.2 Constructor & Destructor Documentation

### 5.54.2.1 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.

## 5.54.3 Member Function Documentation

### 5.54.3.1 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:

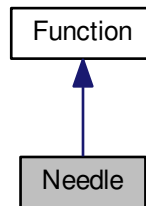
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

## 5.55 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.*

#### 5.55.1 Detailed Description

[Needle](#) in a haystack.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 129 of file theory.hh.

## 5.55.2 Member Function Documentation

### 5.55.2.1 `get_maximum()`

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

Get the global maximum.

#### Returns

1

Reimplemented from [Function](#).

Definition at line 153 of file theory.hh.

### 5.55.2.2 `has_known_maximum()`

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

Check for a known maximum.

#### Returns

true

Reimplemented from [Function](#).

Definition at line 149 of file theory.hh.

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

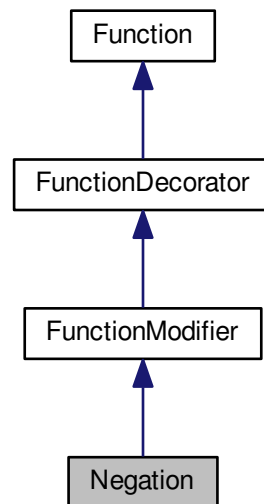
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.56 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

### 5.56.1 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.

### 5.56.2 Member Function Documentation

#### 5.56.2.1 `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.

#### 5.56.2.2 `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.

### 5.56.2.3 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:

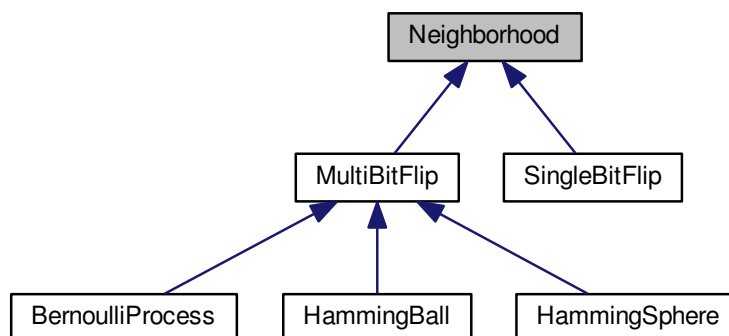
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

## 5.57 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.*

### 5.57.1 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`.

### 5.57.2 Constructor & Destructor Documentation

#### 5.57.2.1 Neighborhood()

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

Constructor.

#### Parameters

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

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

### 5.57.3 Member Function Documentation

#### 5.57.3.1 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.

## 5.57.3.2 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:

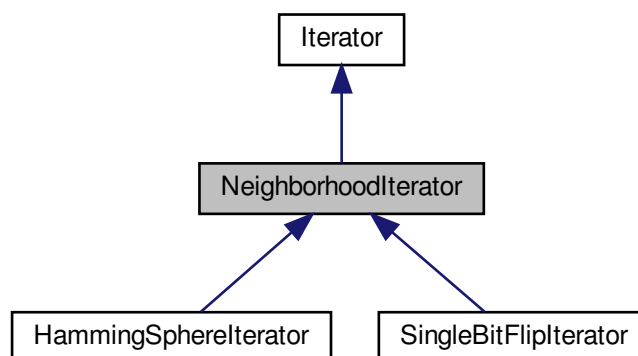
- lib/hnco/neighborhoods/neighborhood.hh

## 5.58 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

### 5.58.1 Detailed Description

[Neighborhood](#) iterator.

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

### 5.58.2 Constructor & Destructor Documentation

#### 5.58.2.1 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:

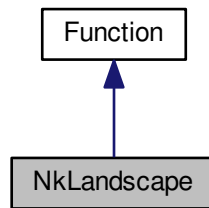
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

## 5.59 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`

#### 5.59.1 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`.

## 5.59.2 Member Function Documentation

### 5.59.2.1 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:

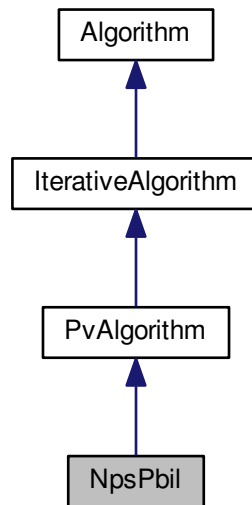
- lib/hnco/functions/nk-landscape.hh
- lib/hnco/functions/nk-landscape.cc

## 5.60 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\\_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 _rate = 1e-3`  
*Learning rate.*

### Additional Inherited Members

#### 5.60.1 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`

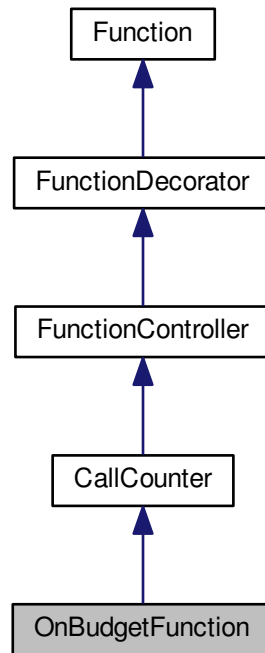
## 5.61 OnBudgetFunction Class Reference

[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

### 5.61.1 Detailed Description

[CallCounter](#) with a limited number of evaluations.

Definition at line 310 of file function-controller.hh.

### 5.61.2 Member Function Documentation

#### 5.61.2.1 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.

#### 5.61.2.2 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.

## 5.61.2.3 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:

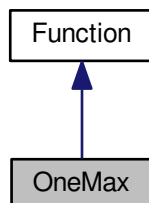
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.62 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.*

## 5.62.1 Detailed Description

[OneMax](#).

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 36 of file theory.hh.

## 5.62.2 Member Function Documentation

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

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

Get the global maximum.

Returns

[\\_bv\\_size](#)

Reimplemented from [Function](#).

Definition at line 57 of file theory.hh.

#### 5.62.2.2 has\_known\_maximum()

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

Check for a known maximum.

##### Returns

true

Reimplemented from [Function](#).

Definition at line 61 of file theory.hh.

#### 5.62.2.3 provides\_incremental\_evaluation()

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

Check whether the function provides incremental evaluation.

##### Returns

true

Reimplemented from [Function](#).

Definition at line 66 of file theory.hh.

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

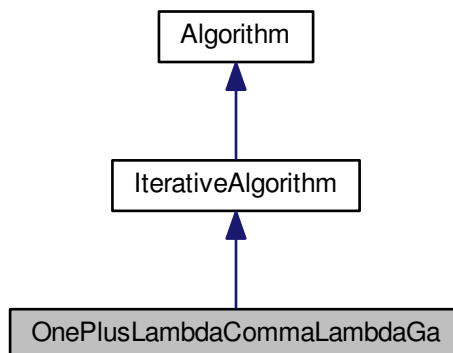
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.63 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

### 5.63.1 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.

### 5.63.2 Constructor & Destructor Documentation

#### 5.63.2.1 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:

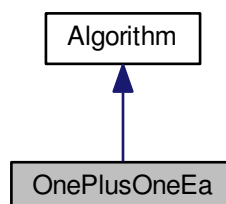
- `lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh`
- `lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc`

## 5.64 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

### 5.64.1 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](#).

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 44 of file one-plus-one-ea.hh.



## 5.64.2 Constructor & Destructor Documentation

### 5.64.2.1 OnePlusOneEa()

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

Constructor.

#### Parameters

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

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

Definition at line 79 of file `one-plus-one-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 125 of file `one-plus-one-ea.hh`.

### 5.64.3.2 set\_num\_iterations()

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

Set the number of iterations.

#### Parameters

<i>x</i>	Number of iterations
----------	----------------------

$x \leq 0$  means indefinite

Definition at line 115 of file one-plus-one-ea.hh.

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

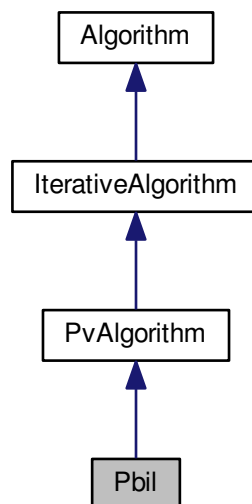
- lib/hnco/algorithms/ea/one-plus-one-ea.hh

## 5.65 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\_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 `_rate` = 1e-3  
*Learning rate.*

### Additional Inherited Members

#### 5.65.1 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:

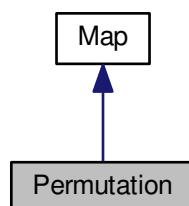
- `lib/hnco/algorithms/pv/pbil.hh`
- `lib/hnco/algorithms/pv/pbil.cc`

## 5.66 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**

### 5.66.1 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.

### 5.66.2 Member Function Documentation

### 5.66.2.1 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:

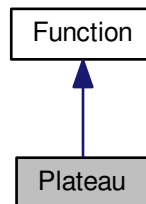
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.67 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.*

### 5.67.1 Detailed Description

[Plateau](#).

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 239 of file theory.hh.

### 5.67.2 Member Function Documentation

#### 5.67.2.1 `get_maximum()`

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

Get the global maximum.

Returns

`_bv_size + 2`

Reimplemented from [Function](#).

Definition at line 263 of file theory.hh.

#### 5.67.2.2 `has_known_maximum()`

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

Check for a known maximum.

Returns

`true`

Reimplemented from [Function](#).

Definition at line 259 of file theory.hh.

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

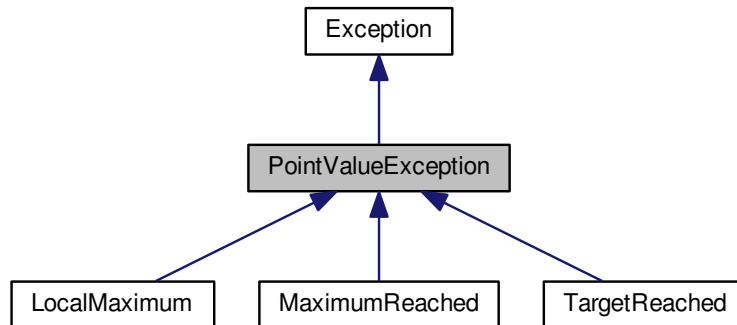
- `lib/hnco/functions/theory.hh`
- `lib/hnco/functions/theory.cc`

## 5.68 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.*

#### 5.68.1 Detailed Description

Point-value exception.

Definition at line 38 of file exception.hh.

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

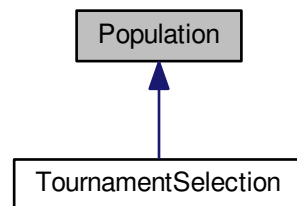
- lib/hnco/exception.hh

## 5.69 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.*
- [bit\\_vector\\_t](#) & [get\\_bv](#) (int i)  
*Get a bit vector.*
- const [bit\\_vector\\_t](#) & [get\\_bv](#) (int i) const  
*Get a bit vector.*

#### Get sorted bit vectors

- const [bit\\_vector\\_t](#) & [get\\_best\\_bv](#) (int i) const  
*Get best bit vector.*
- const [bit\\_vector\\_t](#) & [get\\_best\\_bv](#) () 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.*

### Selection

- void [plus\\_selection](#) (const [Population](#) &offsprings)  
*Plus selection.*
- void [comma\\_selection](#) (const [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.*

## 5.69.1 Detailed Description

[Population](#).

Definition at line 36 of file `population.hh`.

## 5.69.2 Member Function Documentation

### 5.69.2.1 [comma\\_selection\(\)](#)

```
void comma_selection (
    const Population & offsprings )
```

Comma selection.

#### Precondition

Offspring population must be sorted.

#### Warning

The function does not break ties randomly as it should.

Definition at line 93 of file `population.cc`.

#### 5.69.2.2 `get_best_bv()` [1/2]

```
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 90 of file `population.hh`.

#### 5.69.2.3 `get_best_bv()` [2/2]

```
const bit\_vector\_t& get_best_bv ( ) const [inline]
```

Get best bit vector.

##### Precondition

The population must be sorted.

Definition at line 96 of file `population.hh`.

#### 5.69.2.4 `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 119 of file `population.hh`.

#### 5.69.2.5 `get_best_value()` [2/2]

```
double get_best_value ( ) const [inline]
```

Get best value.

##### Precondition

The population must be sorted.

Definition at line 125 of file `population.hh`.

#### 5.69.2.6 `get_worst_bv()`

```
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 104 of file `population.hh`.

#### 5.69.2.7 `plus_selection()`

```
void plus_selection (
    const Population & offsprings )
```

Plus selection.

##### Precondition

Both populations must be sorted.

##### Warning

The function does not break ties randomly as it should.

Definition at line 74 of file `population.cc`.

### 5.69.3 Member Data Documentation

#### 5.69.3.1 `_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`.

#### 5.69.3.2 `_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:

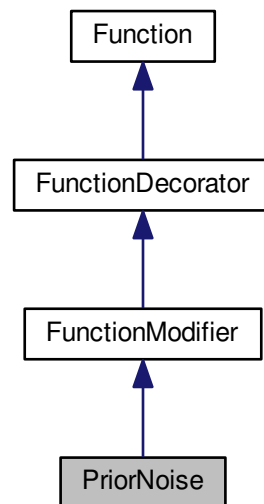
- `lib/hnco/algorithms/population.hh`
- `lib/hnco/algorithms/population.cc`

## 5.70 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

### 5.70.1 Detailed Description

Prior noise.

Definition at line 35 of file prior-noise.hh.

### 5.70.2 Member Function Documentation

#### 5.70.2.1 `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.

#### 5.70.2.2 `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.

### 5.70.2.3 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:

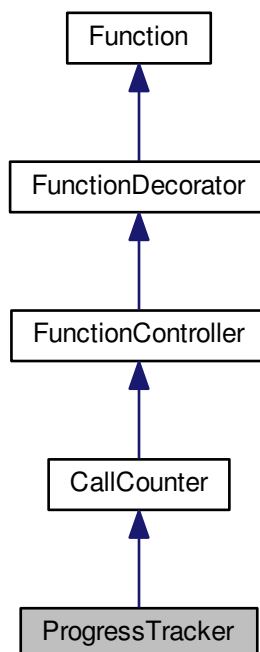
- lib/hnco/functions/decorators/prior-noise.hh
- lib/hnco/functions/decorators/prior-noise.cc

## 5.71 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.*
- const [Event](#) & [get\\_last\\_improvement](#) ()  
*Get the last improvement.*

## 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.*

## 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.*

## Parameters

- bool [\\_log\\_improvement](#) = false  
*Log improvement.*
- std::ostream \* [\\_stream](#) = &std::cout  
*Output stream.*

### 5.71.1 Detailed Description

[ProgressTracker](#).

A [ProgressTracker](#) is a [CallCounter](#) which records the last event, that is the time and value of the last improvement.

Definition at line 212 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>MaximumReached</i>	
<i>TargetReached</i>	

Reimplemented from [CallCounter](#).

Definition at line 153 of file function-controller.cc.

### 5.71.2.2 get\_last\_improvement()

```
const Event& get_last_improvement ( ) [inline]
```

Get the last improvement.

#### Warning

If `_last_improvement.time` is zero then `_function` has never been called. The [Event](#) returned by `get_last_improvement` has therefore no meaning.

Definition at line 288 of file function-controller.hh.

### 5.71.2.3 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 172 of file function-controller.cc.

#### 5.71.2.4 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 191 of file function-controller.cc.

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

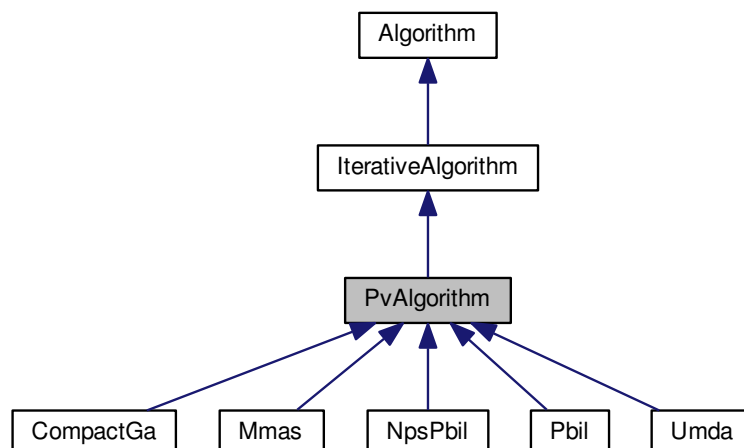
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.72 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.*

### 5.72.1 Detailed Description

Probability vector algorithm.

Definition at line 34 of file pv-algorithm.hh.

### 5.72.2 Member Enumeration Documentation

#### 5.72.2.1 anonymous enum

anonymous enum

## Enumerator

LOG_PV	Log probability vector.
LOG_ENTROPY	Log entropy.

Definition at line 39 of file pv-algorithm.hh.

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

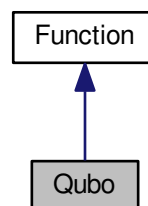
- lib/hnco/algorithms/pv/pv-algorithm.hh
- lib/hnco/algorithms/pv/pv-algorithm.cc

## 5.73 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.*

### 5.73.1 Detailed Description

Quadratic unconstrained binary optimization.

Its expression is of the form  $f(x) = \sum_i Q_{ii}x_i + \sum_{i < j} Q_{ij}x_ix_j = x^T Qx$ , 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.

### 5.73.2 Member Function Documentation

#### 5.73.2.1 load()

```
void load (
    std::istream & stream )
```

Load an instance.

Exceptions

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

Definition at line 35 of file qubo.cc.

### 5.73.3 Member Data Documentation

#### 5.73.3.1 `_q`

```
std::vector<std::vector<double> > _q [private]
```

Matrix.

$n \times n$  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

## 5.74 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 [random\\_bit](#) ()  
*Next random bit.*

### Static Public Attributes

- static std::mt19937 [engine](#)  
*Engine.*

#### 5.74.1 Detailed Description

[Random](#) numbers.

Definition at line 33 of file random.hh.

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

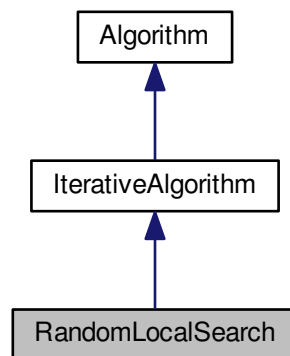
- lib/hnco/random.hh
- lib/hnco/random.cc

## 5.75 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.*

### 5.75.1 Detailed Description

Random local search.

Definition at line 39 of file random-local-search.hh.

### 5.75.2 Member Function Documentation

#### 5.75.2.1 `set_patience()`

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

Set patience.

Number of consecutive rejected moves before throwing a LocalMaximum exception

#### Parameters

<code>x</code>	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:

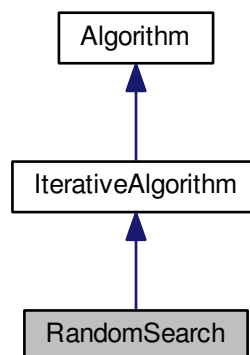
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

## 5.76 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

### 5.76.1 Detailed Description

Random search.

Definition at line 30 of file random-search.hh.

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

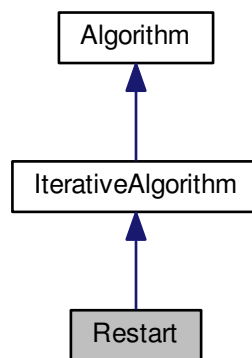
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

## 5.77 Restart Class Reference

[Restart](#).

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

Inheritance diagram for Restart:



## Public Member Functions

- [Restart](#) (int n, [Algorithm](#) \*algorithm)  
*Constructor.*
- 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

#### 5.77.1 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:

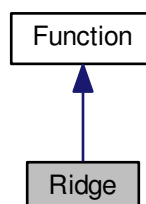
- `lib/hnco/algorithms/restart.hh`
- `lib/hnco/algorithms/restart.cc`

## 5.78 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.*

### 5.78.1 Detailed Description

[Ridge](#).

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 203 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

$2 * \_bv\_size$

Reimplemented from [Function](#).

Definition at line 227 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 223 of file theory.hh.

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

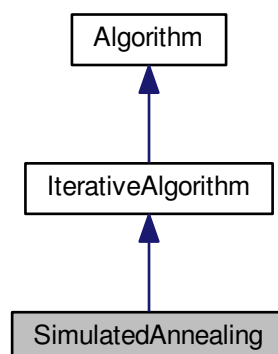
- lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.79 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

### 5.79.1 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.

### 5.79.2 Member Function Documentation

#### 5.79.2.1 init\_beta()

```
void init_beta ( ) [private]
```

Initialize beta.

Requires (2 \* \_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:

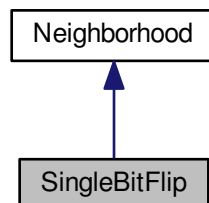
- lib/hnco/algorithms/ls/simulated-annealing.hh
- lib/hnco/algorithms/ls/simulated-annealing.cc

## 5.80 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

### 5.80.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

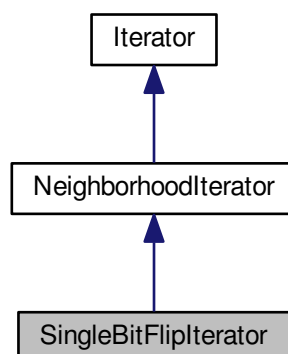
- lib/hnco/neighborhoods/neighborhood.hh

## 5.81 SingleBitFlplerator Class Reference

Single bit flip neighborhood iterator.

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

Inheritance diagram for SingleBitFlplerator:





## Public Member Functions

- [SingleBitFlipIterator](#) (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

### 5.81.1 Detailed Description

Single bit flip neighborhood iterator.

Definition at line 53 of file neighborhood-iterator.hh.

### 5.81.2 Constructor & Destructor Documentation

#### 5.81.2.1 SingleBitFlipIterator()

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

Constructor.

#### Parameters

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

Definition at line 65 of file neighborhood-iterator.hh.

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

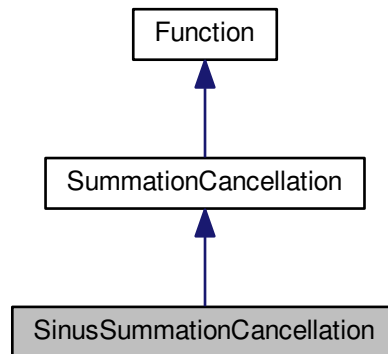
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

## 5.82 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

#### 5.82.1 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:

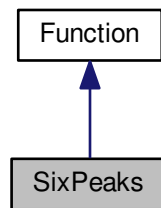
- lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

## 5.83 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.*

### 5.83.1 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.

### 5.83.2 Member Function Documentation

#### 5.83.2.1 `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.

## 5.83.2.2 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

## 5.84 SpinHerding Class Reference

Herding with spin variables.

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

## Public Types

- enum {  
    [SAMPLE\\_GREEDY](#), [SAMPLE\\_RLS](#), [SAMPLE\\_DLS](#), [SAMPLE\\_NN](#),  
    [LAST\\_SAMPLE](#) }

## Public Member Functions

- [SpinHerding](#) (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.*
- double [delta](#) (const [SpinMoment](#) &target)  
    *Compute the norm of the moment increment.*

## 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\\_num\\_par\\_updates](#) (int x)  
    *Set the number of parallel 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.*
- void `sample_nn` (`bit_vector_t` &x)  
*Sample by means of a neural network.*
- void `update_counters` (const `bit_vector_t` &x)  
*Update counters.*

## Protected Attributes

- `SpinMoment_delta`  
*Delta moment.*
- `SpinMoment_count`  
*Counter moment.*
- `bit_vector_t_state`  
*State.*
- `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.*
- `int _num_par_updates` = 1  
*Number of parallel updates per sample.*
- `double _weight` = 1  
*Weight of second order moments.*

### 5.84.1 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 herding-spin.hh.

## 5.84.2 Member Enumeration Documentation

### 5.84.2.1 anonymous enum

anonymous enum

#### Enumerator

SAMPLE_GREEDY	Greedy algorithm.
SAMPLE_RLS	Random local search.
SAMPLE_DLS	Deterministic local search.
SAMPLE_NN	Neural network.

Definition at line 109 of file herding-spin.hh.

## 5.84.3 Constructor & Destructor Documentation

### 5.84.3.1 SpinHerdng()

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

Constructor.

#### Parameters

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

\_num\_seq\_updates is initialized to n.

Definition at line 131 of file herding-spin.hh.

## 5.84.4 Member Function Documentation

### 5.84.4.1 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 155 of file herding-spin.cc.

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

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

## 5.85 SpinMoment Struct Reference

Moment for spin variables.

```
#include <hnco/algorithms/hea/moment-spin.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.*

### Public Attributes

- [std::vector< double > \\_first](#)  
*First moment.*
- [std::vector< std::vector< double > > \\_second](#)  
*Second moment.*
- double [\\_weight](#) = 1  
*Weight of second order moments.*



### 5.85.1 Detailed Description

Moment for spin variables.

Definition at line 35 of file moment-spin.hh.

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

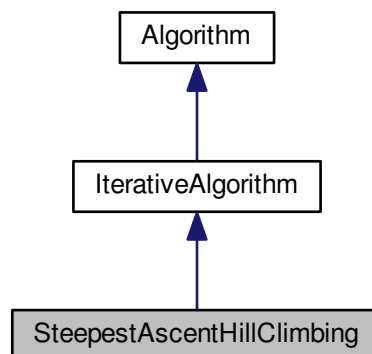
- lib/hnco/algorithms/hea/moment-spin.hh
- lib/hnco/algorithms/hea/moment-spin.cc

## 5.86 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.*

### 5.86.1 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:

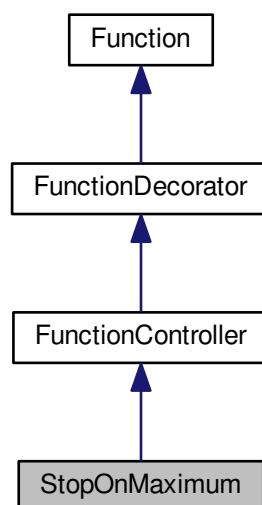
- `lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh`
- `lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc`

## 5.87 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

### 5.87.1 Detailed Description

Stop on maximum.

The [eval\(\)](#) member function throws a MaximumReached exception when its argument maximizes the decorated function.

Definition at line 98 of file function-controller.hh.

### 5.87.2 Constructor & Destructor Documentation

#### 5.87.2.1 StopOnMaximum()

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

Constructor.

#### Parameters

<i>function</i>	Decorated function
-----------------	--------------------

#### Precondition

function->[has\\_known\\_maximum\(\)](#)

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

### 5.87.3 Member Function Documentation

#### 5.87.3.1 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.

#### 5.87.3.2 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.

#### 5.87.3.3 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:

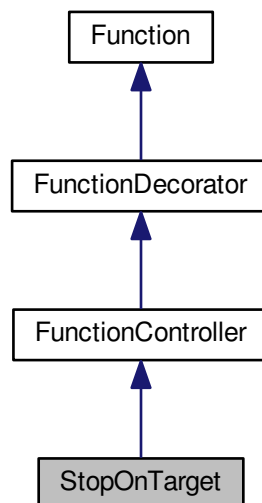
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.88 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,  $\leftrightarrow$  \_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

### 5.88.1 Detailed Description

Stop on target.

Definition at line 134 of file function-controller.hh.

### 5.88.2 Constructor & Destructor Documentation

#### 5.88.2.1 StopOnTarget()

```
StopOnTarget (
    Function * function,
    double target ) [inline]
```

Constructor.

#### Parameters

<i>function</i>	Decorated function
-----------------	--------------------

Definition at line 144 of file function-controller.hh.

### 5.88.3 Member Function Documentation

#### 5.88.3.1 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.

#### 5.88.3.2 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.

#### 5.88.3.3 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:

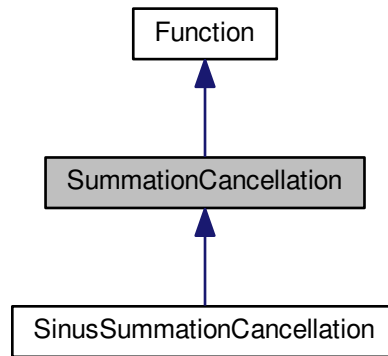
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.89 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.*



### 5.89.1 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.

### 5.89.2 Constructor & Destructor Documentation

#### 5.89.2.1 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

$n$	Size of the bit vector
-----	------------------------

Definition at line 70 of file cancellation.hh.

### 5.89.3 Member Function Documentation

#### 5.89.3.1 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:

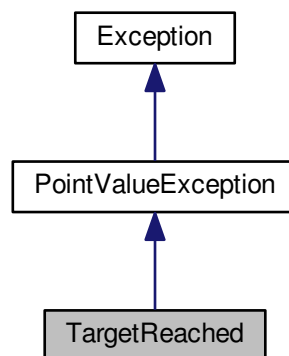
- lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

## 5.90 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**

### 5.90.1 Detailed Description

target reached

Definition at line 61 of file exception.hh.

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

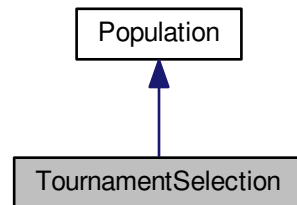
- lib/hnco/exception.hh

## 5.91 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

#### 5.91.1 Detailed Description

[Population](#) with tournament selection.

Definition at line 34 of file tournament-selection.hh.

## 5.91.2 Member Function Documentation

### 5.91.2.1 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:

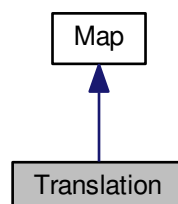
- lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

## 5.92 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**

### 5.92.1 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.

### 5.92.2 Member Function Documentation

### 5.92.2.1 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:

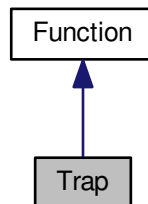
- lib/hnco/map.hh
- lib/hnco/map.cc

## 5.93 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.*

### 5.93.1 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.

### 5.93.2 Constructor & Destructor Documentation

#### 5.93.2.1 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.

### 5.93.3 Member Function Documentation

#### 5.93.3.1 `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.

#### 5.93.3.2 `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`

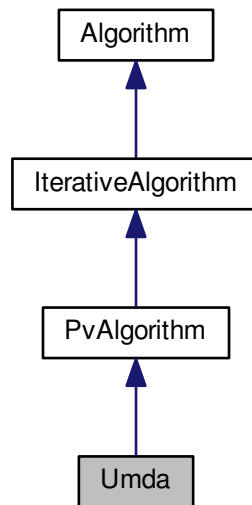
## 5.94 Umda Class Reference

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

### 5.94.1 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:

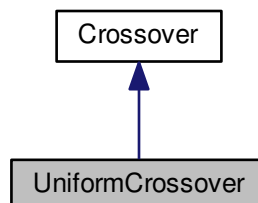
- `lib/hnco/algorithms/pv/umda.hh`
- `lib/hnco/algorithms/pv/umda.cc`

## 5.95 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.*

### 5.95.1 Detailed Description

Uniform crossover.

Definition at line 56 of file `crossover.hh`.

## 5.95.2 Member Function Documentation

### 5.95.2.1 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:

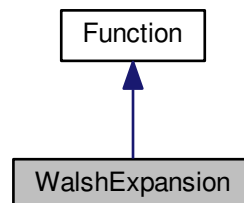
- `lib/hnco/algorithms/ea/crossover.hh`
- `lib/hnco/algorithms/ea/crossover.cc`

## 5.96 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`**

### 5.96.1 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.

### 5.96.2 Member Function Documentation

#### 5.96.2.1 [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:

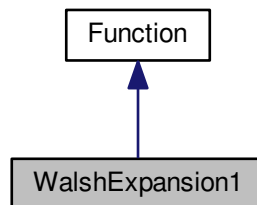
- lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

## 5.97 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`

### 5.97.1 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.

### 5.97.2 Member Function Documentation

#### 5.97.2.1 `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:

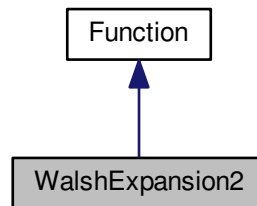
- `lib/hnco/functions/walsh/walsh-expansion-1.hh`
- `lib/hnco/functions/walsh/walsh-expansion-1.cc`

## 5.98 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**

### 5.98.1 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.

### 5.98.2 Member Function Documentation

#### 5.98.2.1 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.

### 5.98.3 Member Data Documentation

#### 5.98.3.1 \_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

## 5.99 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.*

### 5.99.1 Detailed Description

Walsh transform term.

Definition at line 44 of file function.hh.

### 5.99.2 Member Data Documentation

#### 5.99.2.1 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 51 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`, 154
  - `_expression`
    - `hnco::function::MaxSat`, 111
  - `_functions`
    - `hnco::algorithm::Algorithm`, 33
  - `_lookup`
    - `hnco::algorithm::Population`, 154
  - `_q`
    - `hnco::function::Qubo`, 164
  - `_quadratic`
    - `hnco::function::WalshExpansion2`, 206
- AdditiveGaussianNoise, 27
- AffineMap, 29
- Algorithm, 31
- 
- bernoulli\_trials
  - `hnco::neighborhood::MultiBitFlip`, 118
- BernoulliProcess, 33
  - `hnco::neighborhood::BernoulliProcess`, 34, 35
- BiasedCrossover, 36
- BinaryHerdling, 37
- BinaryMoment, 39
- bit\_t
  - `hnco`, 14
- bm\_add\_rows
  - `hnco`, 15
- bm\_identity
  - `hnco`, 15
- bm\_invert
  - `hnco`, 16
- bm\_multiply
  - `hnco`, 16
- bm\_rank
  - `hnco`, 17
- bm\_row\_echelon\_form
  - `hnco`, 17
- bm\_solve
  - `hnco`, 17
- bm\_solve\_upper\_triangular
  - `hnco`, 18
- BmPbil, 40
- breed
  - `hnco::algorithm::BiasedCrossover`, 36
  - `hnco::algorithm::Crossover`, 51
  - `hnco::algorithm::UniformCrossover`, 201
- bv\_from\_vector\_bool
  - `hnco`, 19
- bv\_to\_vector\_bool
  - `hnco`, 19
- 
- Cache, 44
  - `hnco::function::Cache`, 45
- CallCounter, 46
- comma\_selection
  - `hnco::algorithm::Population`, 151
- CompactGa, 47
- CompleteSearch, 49
- compute\_walsh\_transform
  - `hnco::function::Function`, 64
- Crossover, 50
- 
- DeceptiveJump, 51
- 
- EqualProducts, 53
- Error, 55
- eval
  - `hnco::function::OnBudgetFunction`, 136
  - `hnco::function::ProgressTracker`, 159
  - `hnco::function::StopOnMaximum`, 186
  - `hnco::function::StopOnTarget`, 188
- Exception, 56
- 
- Factorization, 57
  - `hnco::function::Factorization`, 58
- feature
  - `hnco::function::Function::WalshTransformTerm`, 207
- FirstAscentHillClimbing, 59
- FourPeaks, 60
- Function, 63
- Function::WalshTransformTerm, 207
- FunctionController, 67
- FunctionDecorator, 68
- FunctionMapComposition, 69
  - `hnco::function::FunctionMapComposition`, 70
- FunctionModifier, 72
- FunctionPlugin, 73
  - `hnco::function::FunctionPlugin`, 74
- 
- GeneticAlgorithm, 74
  - `hnco::algorithm::GeneticAlgorithm`, 76
- get\_best\_bv
  - `hnco::algorithm::Population`, 151, 152
- get\_best\_value
  - `hnco::algorithm::Population`, 152
- get\_last\_improvement
  - `hnco::function::ProgressTracker`, 159
- get\_maximum
  - `hnco::function::AdditiveGaussianNoise`, 28

- hnco::function::DeceptiveJump, 52
  - hnco::function::FourPeaks, 62
  - hnco::function::Function, 65
  - hnco::function::FunctionMapComposition, 71
  - hnco::function::Hiff, 88
  - hnco::function::Jump, 95
  - hnco::function::LeadingOnes, 98
  - hnco::function::Needle, 123
  - hnco::function::Negation, 125
  - hnco::function::OneMax, 138
  - hnco::function::Plateau, 148
  - hnco::function::PriorNoise, 156
  - hnco::function::Ridge, 170
  - hnco::function::SixPeaks, 178
  - hnco::function::Trap, 197
- get\_worst\_bv
  - hnco::algorithm::Population, 153
- HammingBall, 77
  - hnco::neighborhood::HammingBall, 78
- HammingSphere, 79
  - hnco::neighborhood::HammingSphere, 80
- HammingSphereIterator, 80
  - hnco::neighborhood::HammingSphereIterator, 82
- has\_known\_maximum
  - hnco::function::AdditiveGaussianNoise, 28
  - hnco::function::DeceptiveJump, 52
  - hnco::function::FourPeaks, 62
  - hnco::function::FunctionMapComposition, 71
  - hnco::function::Hiff, 88
  - hnco::function::Jump, 95
  - hnco::function::LeadingOnes, 98
  - hnco::function::LinearFunction, 100
  - hnco::function::Needle, 123
  - hnco::function::Negation, 125
  - hnco::function::OneMax, 138
  - hnco::function::Plateau, 148
  - hnco::function::PriorNoise, 156
  - hnco::function::Ridge, 170
  - hnco::function::SixPeaks, 178
  - hnco::function::SummationCancellation, 191
  - hnco::function::Trap, 198
- Hea
  - hnco::algorithm::hea::Hea, 85
- Hea< Moment, Herding >, 82
- Hiff, 87
- hnco, 11
  - bit\_t, 14
  - bm\_add\_rows, 15
  - bm\_identity, 15
  - bm\_invert, 16
  - bm\_multiply, 16
  - bm\_rank, 17
  - bm\_row\_echelon\_form, 17
  - bm\_solve, 17
  - bm\_solve\_upper\_triangular, 18
  - bv\_from\_vector\_bool, 19
  - bv\_to\_vector\_bool, 19
  - sbm\_multiply, 19
  - sparse\_bit\_matrix\_t, 14
  - sparse\_bit\_vector\_t, 15
- hnco::AffineMap
  - is\_surjective, 30
  - random, 30
- hnco::LinearMap
  - is\_surjective, 102
  - random, 102
- hnco::Map
  - is\_surjective, 106
- hnco::MapComposition
  - is\_surjective, 108
  - MapComposition, 107
- hnco::Permutation
  - is\_surjective, 146
- hnco::Translation
  - is\_surjective, 195
- hnco::algorithm, 20
- hnco::algorithm::Algorithm
  - \_functions, 33
- hnco::algorithm::BiasedCrossover
  - breed, 36
- hnco::algorithm::Crossover
  - breed, 51
- hnco::algorithm::GeneticAlgorithm
  - GeneticAlgorithm, 76
  - set\_allow\_stay, 77
- hnco::algorithm::IterativeAlgorithm
  - IterativeAlgorithm, 91
  - maximize, 91
  - set\_num\_iterations, 92
- hnco::algorithm::MuCommaLambdaEa
  - MuCommaLambdaEa, 116
  - set\_allow\_stay, 117
- hnco::algorithm::MuPlusLambdaEa
  - MuPlusLambdaEa, 121
  - set\_allow\_stay, 121
- hnco::algorithm::OnePlusLambdaCommaLambdaGa
  - OnePlusLambdaCommaLambdaGa, 141
- hnco::algorithm::OnePlusOneEa
  - OnePlusOneEa, 143
  - set\_allow\_stay, 143
  - set\_num\_iterations, 143
- hnco::algorithm::Population
  - \_compare\_index\_value, 154
  - \_lookup, 154
  - comma\_selection, 151
  - get\_best\_bv, 151, 152
  - get\_best\_value, 152
  - get\_worst\_bv, 153
  - plus\_selection, 153
- hnco::algorithm::RandomLocalSearch
  - set\_patience, 166
- hnco::algorithm::SimulatedAnnealing
  - init\_beta, 173
- hnco::algorithm::TournamentSelection
  - select, 194
- hnco::algorithm::UniformCrossover

- breed, 201
- hnco::algorithm::bm\_pbil, 22
- hnco::algorithm::bm\_pbil::BmPbil
  - set\_selection\_size, 43
- hnco::algorithm::hea, 22
- hnco::algorithm::hea::Hea
  - Hea, 85
  - set\_reset\_period, 86
  - set\_selection\_size, 86
- hnco::algorithm::hea::SpinHerding
  - q\_variation, 181
  - SpinHerding, 181
- hnco::exception, 23
- hnco::function, 23
- hnco::function::AdditiveGaussianNoise
  - get\_maximum, 28
  - has\_known\_maximum, 28
- hnco::function::Cache
  - Cache, 45
  - provides\_incremental\_evaluation, 45
- hnco::function::DeceptiveJump
  - get\_maximum, 52
  - has\_known\_maximum, 52
- hnco::function::EqualProducts
  - random, 54
- hnco::function::Factorization
  - Factorization, 58
- hnco::function::FourPeaks
  - get\_maximum, 62
  - has\_known\_maximum, 62
- hnco::function::Function
  - compute\_walsh\_transform, 64
  - get\_maximum, 65
  - incremental\_eval, 65
  - provides\_incremental\_evaluation, 66
  - safe\_eval, 66
- hnco::function::Function::WalshTransformTerm
  - feature, 207
- hnco::function::FunctionController
  - provides\_incremental\_evaluation, 68
- hnco::function::FunctionMapComposition
  - FunctionMapComposition, 70
  - get\_maximum, 71
  - has\_known\_maximum, 71
- hnco::function::FunctionPlugin
  - FunctionPlugin, 74
- hnco::function::Hiff
  - get\_maximum, 88
  - has\_known\_maximum, 88
- hnco::function::Jump
  - get\_maximum, 95
  - has\_known\_maximum, 95
- hnco::function::LeadingOnes
  - get\_maximum, 98
  - has\_known\_maximum, 98
- hnco::function::LinearFunction
  - has\_known\_maximum, 100
  - random, 100
- hnco::function::MaxSat
  - \_expression, 111
  - load, 110
  - random, 110, 111
- hnco::function::Needle
  - get\_maximum, 123
  - has\_known\_maximum, 123
- hnco::function::Negation
  - get\_maximum, 125
  - has\_known\_maximum, 125
  - provides\_incremental\_evaluation, 125
- hnco::function::NkLandscape
  - random, 132
- hnco::function::OnBudgetFunction
  - eval, 136
  - incremental\_eval, 136
  - update, 136
- hnco::function::OneMax
  - get\_maximum, 138
  - has\_known\_maximum, 138
  - provides\_incremental\_evaluation, 139
- hnco::function::Plateau
  - get\_maximum, 148
  - has\_known\_maximum, 148
- hnco::function::PriorNoise
  - get\_maximum, 156
  - has\_known\_maximum, 156
  - provides\_incremental\_evaluation, 156
- hnco::function::ProgressTracker
  - eval, 159
  - get\_last\_improvement, 159
  - incremental\_eval, 159
  - update, 160
- hnco::function::Qubo
  - \_q, 164
  - load, 163
- hnco::function::Ridge
  - get\_maximum, 170
  - has\_known\_maximum, 170
- hnco::function::SixPeaks
  - get\_maximum, 178
  - has\_known\_maximum, 178
- hnco::function::StopOnMaximum
  - eval, 186
  - incremental\_eval, 186
  - StopOnMaximum, 185
  - update, 186
- hnco::function::StopOnTarget
  - eval, 188
  - incremental\_eval, 189
  - StopOnTarget, 188
  - update, 189
- hnco::function::SummationCancellation
  - has\_known\_maximum, 191
  - SummationCancellation, 191
- hnco::function::Trap
  - get\_maximum, 197
  - has\_known\_maximum, 198

- Trap, 197
- hnco::function::WalshExpansion
  - random, 202
- hnco::function::WalshExpansion1
  - random, 204
- hnco::function::WalshExpansion2
  - \_quadratic, 206
  - random, 206
- hnco::neighborhood, 25
- hnco::neighborhood::BernoulliProcess
  - BernoulliProcess, 34, 35
  - set\_allow\_stay, 35
  - set\_probability, 35
- hnco::neighborhood::HammingBall
  - HammingBall, 78
- hnco::neighborhood::HammingSphere
  - HammingSphere, 80
- hnco::neighborhood::HammingSphereIterator
  - HammingSphereIterator, 82
- hnco::neighborhood::MultiBitFlip
  - bernoulli\_trials, 118
  - MultiBitFlip, 118
  - reservoir\_sampling, 119
- hnco::neighborhood::Neighborhood
  - map, 128
  - mutate, 129
  - Neighborhood, 128
- hnco::neighborhood::NeighborhoodIterator
  - NeighborhoodIterator, 130
- hnco::neighborhood::SingleBitFlipIterator
  - SingleBitFlipIterator, 175
- hnco::random, 26
- HypercubeIterator, 88
- incremental\_eval
  - hnco::function::Function, 65
  - hnco::function::OnBudgetFunction, 136
  - hnco::function::ProgressTracker, 159
  - hnco::function::StopOnMaximum, 186
  - hnco::function::StopOnTarget, 189
- init\_beta
  - hnco::algorithm::SimulatedAnnealing, 173
- is\_surjective
  - hnco::AffineMap, 30
  - hnco::LinearMap, 102
  - hnco::Map, 106
  - hnco::MapComposition, 108
  - hnco::Permutation, 146
  - hnco::Translation, 195
- IterativeAlgorithm, 89
  - hnco::algorithm::IterativeAlgorithm, 91
- Iterator, 92
- Jump, 94
- Labs, 95
- LastEvaluation, 96
- LeadingOnes, 97
- LinearFunction, 99
- LinearMap, 101
- load
  - hnco::function::MaxSat, 110
  - hnco::function::Qubo, 163
- LocalMaximum, 103
- LongPath, 104
- Map, 105
- map
  - hnco::neighborhood::Neighborhood, 128
- MapComposition, 106
  - hnco::MapComposition, 107
- MaxSat, 109
- maximize
  - hnco::algorithm::IterativeAlgorithm, 91
- MaximumReached, 108
- Mmas, 112
- Model, 113
- ModelParameters, 114
- MuCommaLambdaEa, 115
  - hnco::algorithm::MuCommaLambdaEa, 116
- MuPlusLambdaEa, 119
  - hnco::algorithm::MuPlusLambdaEa, 121
- MultiBitFlip, 117
  - hnco::neighborhood::MultiBitFlip, 118
- mutate
  - hnco::neighborhood::Neighborhood, 129
- Needle, 122
- Negation, 124
- Neighborhood, 126
  - hnco::neighborhood::Neighborhood, 128
- NeighborhoodIterator, 129
  - hnco::neighborhood::NeighborhoodIterator, 130
- NkLandscape, 130
- NpsPbil, 132
- OnBudgetFunction, 134
- OneMax, 137
- OnePlusLambdaCommaLambdaGa, 139
  - hnco::algorithm::OnePlusLambdaCommaLambdaGa, 141
- OnePlusOneEa, 141
  - hnco::algorithm::OnePlusOneEa, 143
- Pbil, 144
- Permutation, 145
- Plateau, 147
- plus\_selection
  - hnco::algorithm::Population, 153
- PointValueException, 149
- Population, 150
- PriorNoise, 155
- ProgressTracker, 157
- ProgressTracker::Event, 56
- provides\_incremental\_evaluation
  - hnco::function::Cache, 45
  - hnco::function::Function, 66
  - hnco::function::FunctionController, 68

- hnco::function::Negation, 125
  - hnco::function::OneMax, 139
  - hnco::function::PriorNoise, 156
- PvAlgorithm, 160
- q\_variation
  - hnco::algorithm::hea::SpinHerding, 181
- Qubo, 162
- Random, 164
- random
  - hnco::AffineMap, 30
  - hnco::LinearMap, 102
  - hnco::function::EqualProducts, 54
  - hnco::function::LinearFunction, 100
  - hnco::function::MaxSat, 110, 111
  - hnco::function::NkLandscape, 132
  - hnco::function::WalshExpansion, 202
  - hnco::function::WalshExpansion1, 204
  - hnco::function::WalshExpansion2, 206
- RandomLocalSearch, 165
- RandomSearch, 167
- reservoir\_sampling
  - hnco::neighborhood::MultiBitFlip, 119
- Restart, 168
- Ridge, 169
- safe\_eval
  - hnco::function::Function, 66
- sbm\_multiply
  - hnco, 19
- select
  - hnco::algorithm::TournamentSelection, 194
- set\_allow\_stay
  - hnco::algorithm::GeneticAlgorithm, 77
  - hnco::algorithm::MuCommaLambdaEa, 117
  - hnco::algorithm::MuPlusLambdaEa, 121
  - hnco::algorithm::OnePlusOneEa, 143
  - hnco::neighborhood::BernoulliProcess, 35
- set\_num\_iterations
  - hnco::algorithm::IterativeAlgorithm, 92
  - hnco::algorithm::OnePlusOneEa, 143
- set\_patience
  - hnco::algorithm::RandomLocalSearch, 166
- set\_probability
  - hnco::neighborhood::BernoulliProcess, 35
- set\_reset\_period
  - hnco::algorithm::hea::Hea, 86
- set\_selection\_size
  - hnco::algorithm::bm\_pbil::BmPbil, 43
  - hnco::algorithm::hea::Hea, 86
- SimulatedAnnealing, 171
- SingleBitFlip, 173
- SingleBitFlipIteator, 174
  - hnco::neighborhood::SingleBitFlipIteator, 175
- SinusSummationCancellation, 176
- SixPeaks, 177
- sparse\_bit\_matrix\_t
  - hnco, 14
- sparse\_bit\_vector\_t
  - hnco, 15
- SpinHerding, 179
  - hnco::algorithm::hea::SpinHerding, 181
- SpinMoment, 182
- SteepestAscentHillClimbing, 183
- StopOnMaximum, 184
  - hnco::function::StopOnMaximum, 185
- StopOnTarget, 187
  - hnco::function::StopOnTarget, 188
- SummationCancellation, 189
  - hnco::function::SummationCancellation, 191
- TargetReached, 192
- TournamentSelection, 193
- Translation, 194
- Trap, 196
  - hnco::function::Trap, 197
- Umda, 198
- UniformCrossover, 200
- update
  - hnco::function::OnBudgetFunction, 136
  - hnco::function::ProgressTracker, 160
  - hnco::function::StopOnMaximum, 186
  - hnco::function::StopOnTarget, 189
- WalshExpansion, 201
- WalshExpansion1, 203
- WalshExpansion2, 205