HNCO

0.7

Generated by Doxygen 1.8.13

# **Contents**

1	Nam	espace	Index		1
	1.1	Names	space List		. 1
2	Hier	archica	l Index		3
	2.1	Class I	Hierarchy		. 3
3	Clas	s Index			7
	3.1	Class I	List		. 7
4	Nam	nespace	Docume	ntation	11
	4.1	hnco N	lamespace	e 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	. 14
			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()	. 15
			4.1.3.3	bm_invert()	. 15
			4.1.3.4	bm_multiply()	. 16
			4.1.3.5	bm_solve()	. 16
			4.1.3.6	bm_solve_upper_triangular()	. 17
			4.1.3.7	sbm_multiply()	. 18
	4.2	hnco::a	algorithm N	Namespace Reference	. 18

ii CONTENTS

		4.2.1	Detailed Description	20
	4.3	hnco::a	algorithm::bm_pbil Namespace Reference	20
		4.3.1	Detailed Description	20
	4.4	hnco::a	algorithm::hea Namespace Reference	21
		4.4.1	Detailed Description	21
	4.5	hnco::e	exception Namespace Reference	21
		4.5.1	Detailed Description	21
	4.6	hnco::f	function Namespace Reference	22
		4.6.1	Detailed Description	23
	4.7	hnco::i	neighborhood Namespace Reference	23
		4.7.1	Detailed Description	24
	4.8	hnco::i	random Namespace Reference	24
		4.8.1	Detailed Description	24
5	Clas	s Docu	mentation 2	25
	5.1	Additiv	veGaussianNoise Class Reference	25
		5.1.1	Detailed Description	27
		5.1.2	Member Function Documentation	27
			5.1.2.1 get_maximum()	27
			5.1.2.2 has_known_maximum()	27
	5.2	Affine	Map Class Reference	28
		5.2.1	Detailed Description	29
	5.3	Algorit	hm Class Reference	30
		5.3.1	Detailed Description	31
		5.3.2	Member Data Documentation	31
			5.3.2.1 _functions	31
	5.4	Bernoi	ulliProcess Class Reference	32
		5.4.1	Detailed Description	33
		5.4.2	Constructor & Destructor Documentation	33
			5.4.2.1 BernoulliProcess() [1/2]	33
			5.4.2.2 BernoulliProcess() [2/2]	34

CONTENTS

	5.4.3	Member Function Documentation	34
		5.4.3.1 set_probability()	34
	5.4.4	Member Data Documentation	34
		5.4.4.1 _allow_stay	35
5.5	Biased	Crossover Class Reference	35
	5.5.1	Detailed Description	36
	5.5.2	Member Function Documentation	36
		5.5.2.1 breed()	36
5.6	Binaryl	Herding Class Reference	36
	5.6.1	Detailed Description	38
	5.6.2	Member Enumeration Documentation	38
		5.6.2.1 anonymous enum	38
5.7	Binary	Moment Struct Reference	38
	5.7.1	Detailed Description	39
5.8	BmPbil	Class Reference	39
	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
5.9	Cache	Class Reference	43
	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	46
		5.9.3.1 provides_incremental_evaluation()	46
5.10	CallCo	unter Class Reference	46
	5.10.1	Detailed Description	48
5.11	Compa	ctGa Class Reference	48
	5.11.1	Detailed Description	50

iv CONTENTS

5.12	CompleteSearch Class Reference	 . 51
	5.12.1 Detailed Description	 . 52
5.13	Crossover Class Reference	 . 52
	5.13.1 Detailed Description	 . 53
	5.13.2 Member Function Documentation	 . 53
	5.13.2.1 breed()	 . 53
5.14	DeceptiveJump Class Reference	 . 53
	5.14.1 Detailed Description	 . 55
	5.14.2 Member Function Documentation	 . 55
	5.14.2.1 get_maximum()	 . 55
	5.14.2.2 has_known_maximum()	 . 56
5.15	EqualProducts Class Reference	 . 56
	5.15.1 Detailed Description	 . 58
	5.15.2 Member Function Documentation	 . 58
	5.15.2.1 random()	 . 58
5.16	Error Class Reference	 . 59
	5.16.1 Detailed Description	 . 60
5.17	ProgressTracker::Event Struct Reference	 . 60
	5.17.1 Detailed Description	 . 60
5.18	Exception Class Reference	 . 61
	5.18.1 Detailed Description	 . 61
5.19	Factorization Class Reference	 . 61
	5.19.1 Detailed Description	 . 63
	5.19.2 Constructor & Destructor Documentation	 . 64
	5.19.2.1 Factorization()	 . 64
5.20	FourPeaks Class Reference	 . 64
	5.20.1 Detailed Description	 . 66
	5.20.2 Member Function Documentation	 . 67
	5.20.2.1 get_maximum()	 . 67
	5.20.2.2 has_known_maximum()	 . 67

CONTENTS

5.21	Functio	on Class Reference								
	5.21.1	Detailed Description								
	5.21.2	Member Function Documentation	68							
		5.21.2.1 get_maximum()	68							
		5.21.2.2 incremental_eval()	69							
		5.21.2.3 provides_incremental_evaluation()	69							
		5.21.2.4 safe_eval()	70							
5.22	Functio	nController Class Reference	70							
	5.22.1	Detailed Description	72							
	5.22.2	Member Function Documentation	72							
		5.22.2.1 provides_incremental_evaluation()	72							
5.23	Functio	nDecorator Class Reference	73							
	5.23.1	Detailed Description	74							
5.24	Functio	nMapComposition Class Reference	74							
	5.24.1	Detailed Description	76							
	5.24.2	Constructor & Destructor Documentation	76							
		5.24.2.1 FunctionMapComposition()	76							
	5.24.3	Member Function Documentation	77							
		5.24.3.1 get_maximum()	77							
		5.24.3.2 has_known_maximum()	77							
5.25	Functio	nModifier Class Reference	78							
	5.25.1	Detailed Description	79							
5.26	Functio	nPlugin Class Reference	79							
	5.26.1	Detailed Description	81							
	5.26.2	Constructor & Destructor Documentation	81							
		5.26.2.1 FunctionPlugin()	81							
5.27	Genetic	cAlgorithm Class Reference	82							
	5.27.1	Detailed Description	84							
	5.27.2	Constructor & Destructor Documentation	84							
		5.27.2.1 GeneticAlgorithm()	84							

vi

	5.27.3	Member Function Documentation	84
		5.27.3.1 set_allow_stay()	84
5.28	Hammi	ngBall Class Reference	85
	5.28.1	Detailed Description	87
	5.28.2	Constructor & Destructor Documentation	87
		5.28.2.1 HammingBall()	87
5.29	Hammi	ngBallIterator Class Reference	87
	5.29.1	Detailed Description	89
	5.29.2	Constructor & Destructor Documentation	89
		5.29.2.1 HammingBallIterator()	89
5.30	Hammi	ngSphere Class Reference	89
	5.30.1	Detailed Description	91
	5.30.2	Constructor & Destructor Documentation	91
		5.30.2.1 HammingSphere()	91
5.31	Hea< I	Moment, Herding > Class Template Reference	92
	5.31.1	Detailed Description	94
	5.31.2	Member Enumeration Documentation	94
		5.31.2.1 anonymous enum	94
		5.31.2.2 anonymous enum	95
5.32	Hiff Cla	ss Reference	95
	5.32.1	Detailed Description	97
	5.32.2	Member Function Documentation	97
		5.32.2.1 get_maximum()	97
		5.32.2.2 has_known_maximum()	98
5.33	Hyperc	ubelterator Class Reference	98
	5.33.1	Detailed Description	99
	5.33.2	Member Function Documentation	99
		5.33.2.1 next()	99
5.34	Iterative	eAlgorithm Class Reference	100
	5.34.1	Detailed Description	101

CONTENTS vii

	5.34.2	Constructor & Destructor Documentation	)1
		5.34.2.1 IterativeAlgorithm()	)1
	5.34.3	Member Function Documentation	)1
		5.34.3.1 maximize()	)1
		5.34.3.2 set_num_iterations()	)2
5.35	Iterator	Class Reference	)2
	5.35.1	Detailed Description	)4
5.36	Jump (	Class Reference	)4
	5.36.1	Detailed Description	)6
	5.36.2	Member Function Documentation	)6
		5.36.2.1 get_maximum()	)6
		5.36.2.2 has_known_maximum()	)7
5.37	Labs C	lass Reference	)7
	5.37.1	Detailed Description	)9
5.38	LastEv	aluation Class Reference	)9
	5.38.1	Detailed Description	10
5.39	Leading	gOnes Class Reference	10
	5.39.1	Detailed Description	12
	5.39.2	Member Function Documentation	12
		5.39.2.1 get_maximum()	12
		5.39.2.2 has_known_maximum()	12
5.40	LinearF	Function Class Reference	13
	5.40.1	Detailed Description	14
	5.40.2	Member Function Documentation	14
		5.40.2.1 has_known_maximum()	14
		5.40.2.2 random()	15
5.41	Linear	Map Class Reference	16
	5.41.1	Detailed Description	17
5.42	LocalM	aximum Class Reference	18
	5.42.1	Detailed Description	18

viii CONTENTS

5.43	B LongPath Class Reference	119
	5.43.1 Detailed Description	120
5.44	Map Class Reference	121
	5.44.1 Detailed Description	121
	5.44.2 Member Function Documentation	121
	5.44.2.1 is_surjective()	122
5.45	MapComposition Class Reference	122
	5.45.1 Detailed Description	123
	5.45.2 Constructor & Destructor Documentation	123
	5.45.2.1 MapComposition()	123
	5.45.3 Member Function Documentation	124
	5.45.3.1 is_surjective()	124
5.46	MaximumReached Class Reference	124
	5.46.1 Detailed Description	125
5.47	<sup>7</sup> MaxSat Class Reference	126
	5.47.1 Detailed Description	127
	5.47.2 Member Function Documentation	127
	5.47.2.1 load()	127
	<b>5.47.2.2</b> random() [1/2]	128
	<b>5.47.2.3</b> random() [2/2]	128
	5.47.3 Member Data Documentation	129
	5.47.3.1 _expression	129
5.48	8 Mmas Class Reference	129
	5.48.1 Detailed Description	131
5.49	Model Class Reference	131
	5.49.1 Detailed Description	132
5.50	ModelParameters Class Reference	132
	5.50.1 Detailed Description	133
5.51	MuCommaLambdaEa Class Reference	134
	5.51.1 Detailed Description	135

CONTENTS

	5.51.2	Constructor & Destructor Documentation	136
		5.51.2.1 MuCommaLambdaEa()	136
	5.51.3	Member Function Documentation	136
		5.51.3.1 set_allow_stay()	136
5.52	MultiBit	Flip Class Reference	137
	5.52.1	Detailed Description	138
	5.52.2	Constructor & Destructor Documentation	138
		5.52.2.1 MultiBitFlip()	138
	5.52.3	Member Function Documentation	138
		5.52.3.1 bernoulli_trials()	138
		5.52.3.2 reservoir_sampling()	139
5.53	MuPlus	sLambdaEa Class Reference	139
	5.53.1	Detailed Description	141
	5.53.2	Constructor & Destructor Documentation	141
		5.53.2.1 MuPlusLambdaEa()	141
	5.53.3	Member Function Documentation	142
		5.53.3.1 set_allow_stay()	142
5.54	Needle	Class Reference	142
	5.54.1	Detailed Description	144
	5.54.2	Member Function Documentation	144
		5.54.2.1 get_maximum()	144
		5.54.2.2 has_known_maximum()	144
5.55	Negatio	on Class Reference	145
	5.55.1	Detailed Description	146
	5.55.2	Member Function Documentation	146
		5.55.2.1 get_maximum()	146
		5.55.2.2 has_known_maximum()	147
		5.55.2.3 provides_incremental_evaluation()	147
5.56	Neighb	orhood Class Reference	148
	5.56.1	Detailed Description	149

CONTENTS

	5.56.2	Constructor & Destructor Documentation	50
		5.56.2.1 Neighborhood()	50
	5.56.3	Member Function Documentation	50
		5.56.3.1 map()	50
		5.56.3.2 mutate()	50
5.57	Neighb	porhoodIterator Class Reference	51
	5.57.1	Detailed Description	52
	5.57.2	Constructor & Destructor Documentation	52
		5.57.2.1 NeighborhoodIterator()	52
5.58	NkLand	dscape Class Reference	52
	5.58.1	Detailed Description	54
	5.58.2	Member Function Documentation	54
		5.58.2.1 random()	54
5.59	NpsPbi	il Class Reference	55
	5.59.1	Detailed Description	57
5.60	OnBud	getFunction Class Reference	58
	5.60.1	Detailed Description	59
	5.60.2	Member Function Documentation	59
		5.60.2.1 eval()	59
		5.60.2.2 incremental_eval()	60
		5.60.2.3 update()	60
5.61	OneMa	ax Class Reference	60
	5.61.1	Detailed Description	62
	5.61.2	Member Function Documentation	62
		5.61.2.1 get_maximum()	62
		5.61.2.2 has_known_maximum()	63
		5.61.2.3 provides_incremental_evaluation()	63
5.62	OnePlu	usLambdaCommaLambdaGa Class Reference	63
	5.62.1	Detailed Description	65
	5.62.2	Constructor & Destructor Documentation	65

CONTENTS xi

		5.62.2.1	OnePlus	Lambda	aComm	aLaml	odaGa	ι() .	 	 	 	 		165
5.63	OnePlu	ısOneEa (	Class Ref	erence					 	 	 	 	. <b></b>	166
	5.63.1	Detailed	Description	on					 	 	 	 		168
	5.63.2	Construc	tor & Des	tructor [	Docume	entatio	1		 	 	 	 		168
		5.63.2.1	OnePlus	sOneEa	()				 	 	 	 		168
	5.63.3	Member	Function	Docume	entation				 	 	 	 		169
		5.63.3.1	set_allo	w_stay()					 	 	 	 		169
		5.63.3.2	set_nun	n_iteratio	ons() .				 	 	 	 		169
5.64	Pbil Cla	ass Refere	ence						 	 	 	 		169
	5.64.1	Detailed	Description	on					 	 	 	 		171
5.65	Permut	ation Clas	s Referer	nce					 	 	 	 		172
	5.65.1	Detailed	Description	on					 	 	 	 		173
	5.65.2	Member	Function	Docume	ntation				 	 	 	 		173
		5.65.2.1	is_surje	ctive()					 	 	 	 		173
5.66	Plateau	ı Class Re	eference						 	 	 	 		174
	5.66.1	Detailed	Description	on					 	 	 	 		175
	5.66.2	Member	Function	Docume	ntation				 	 	 	 		175
		5.66.2.1	get_max	kimum()					 	 	 	 		175
		5.66.2.2	has_kno	own_ma	ximum()	)			 	 	 	 		176
5.67	PointVa	alueExcept	tion Class	Refere	nce .				 	 	 	 		176
	5.67.1	Detailed	Description	on					 	 	 	 		177
5.68	Popula	tion Class	Reference	e					 	 	 	 		177
	5.68.1	Detailed	Description	on					 	 	 	 		180
	5.68.2	Member	Function	Docume	entation				 	 	 	 		180
		5.68.2.1	comma	_selectio	on()				 	 	 	 		180
		5.68.2.2	get_bes	t_bv() [	1/2] .				 	 	 	 		180
		5.68.2.3	get_bes	t_bv() [:	2/2] .				 	 	 	 		181
		5.68.2.4	get_bes	t_value(	<b>)</b> [1/2]				 	 	 	 		181
		5.68.2.5	get_bes	t_value(	<b>)</b> [2/2]				 	 	 	 		181
		5.68.2.6	get_wor	st_bv()					 	 	 	 		182

xii CONTENTS

		5.68.2.7 plus_selection()	32
	5.68.3	Member Data Documentation	32
		5.68.3.1 _lookup	32
		5.68.3.2 _operator	33
5.69	PriorNo	pise Class Reference	33
	5.69.1	Detailed Description	35
	5.69.2	Member Function Documentation	35
		5.69.2.1 get_maximum()	35
		5.69.2.2 has_known_maximum()	36
		5.69.2.3 provides_incremental_evaluation()	36
5.70	Progre	ssTracker Class Reference	36
	5.70.1	Detailed Description	38
	5.70.2	Member Function Documentation	38
		5.70.2.1 eval()	38
		5.70.2.2 get_last_improvement()	39
		5.70.2.3 incremental_eval()	39
		5.70.2.4 update()	39
5.71	PvAlgo	rithm Class Reference	90
	5.71.1	Detailed Description	)2
	5.71.2	Member Enumeration Documentation	)2
		5.71.2.1 anonymous enum	)2
5.72	Qubo C	Class Reference	)3
	5.72.1	Detailed Description	)4
	5.72.2	Member Function Documentation	)4
		5.72.2.1 load()	<b>)</b> 4
	5.72.3	Member Data Documentation	<del>)</del> 5
		5.72.3.1 _q	<del>)</del> 5
5.73	Rando	m Struct Reference	<del>)</del> 5
	5.73.1	Detailed Description	<del>)</del> 6
5.74	Randoi	mLocalSearch Class Reference	96

CONTENTS xiii

	5.74.1	Detailed Description	198
	5.74.2	Member Function Documentation	199
		5.74.2.1 set_patience()	199
5.75	Rando	Search Class Reference	199
	5.75.1	Detailed Description	201
5.76	Restar	Class Reference	201
	5.76.1	Detailed Description	203
5.77	Ridge (	lass Reference	203
	5.77.1	Detailed Description	205
	5.77.2	Member Function Documentation	205
		5.77.2.1 get_maximum()	205
		5.77.2.2 has_known_maximum()	205
5.78	Simula	edAnnealing Class Reference	206
	5.78.1	Detailed Description	208
	5.78.2	Member Function Documentation	208
		5.78.2.1 init_beta()	208
5.79	Single	tFlip Class Reference	208
	5.79.1	Detailed Description	209
5.80	Single	tFlipIterator Class Reference	210
	5.80.1	Detailed Description	211
	5.80.2	Constructor & Destructor Documentation	211
		5.80.2.1 SingleBitFlipIterator()	211
5.81	SinusS	mmationCancellation Class Reference	211
	5.81.1	Detailed Description	213
5.82	SixPea	s Class Reference	213
	5.82.1	Detailed Description	215
	5.82.2	Member Function Documentation	216
		5.82.2.1 get_maximum()	216
		5.82.2.2 has_known_maximum()	216
5.83	SpinHe	ding Class Reference	216

xiv CONTENTS

	5.83.1	Detailed Description	218
	5.83.2	Member Enumeration Documentation	218
		5.83.2.1 anonymous enum	218
	5.83.3	Member Function Documentation	218
		5.83.3.1 q_variation()	219
5.84	SpinMo	oment Struct Reference	219
	5.84.1	Detailed Description	220
5.85	Steepe	stAscentHillClimbing Class Reference	220
	5.85.1	Detailed Description	222
5.86	StopOr	Maximum Class Reference	222
	5.86.1	Detailed Description	224
	5.86.2	Constructor & Destructor Documentation	224
		5.86.2.1 StopOnMaximum()	224
	5.86.3	Member Function Documentation	224
		5.86.3.1 eval()	224
		5.86.3.2 incremental_eval()	225
		5.86.3.3 update()	225
5.87	StopOr	Target Class Reference	226
	5.87.1	Detailed Description	227
	5.87.2	Constructor & Destructor Documentation	227
		5.87.2.1 StopOnTarget()	227
	5.87.3	Member Function Documentation	228
		5.87.3.1 eval()	228
		5.87.3.2 incremental_eval()	228
		5.87.3.3 update()	228
5.88	Summa	ationCancellation Class Reference	229
	5.88.1	Detailed Description	231
	5.88.2	Constructor & Destructor Documentation	231
		5.88.2.1 SummationCancellation()	231
	5.88.3	Member Function Documentation	232

CONTENTS xv

	5.88.3.1 has_known_maximum()	232
5.89	TargetReached Class Reference	232
	5.89.1 Detailed Description	233
5.90	TournamentSelection Class Reference	234
	5.90.1 Detailed Description	235
	5.90.2 Member Function Documentation	235
	5.90.2.1 select()	235
5.91	Translation Class Reference	236
	5.91.1 Detailed Description	237
	5.91.2 Member Function Documentation	237
	5.91.2.1 is_surjective()	237
5.92	Trap Class Reference	238
	5.92.1 Detailed Description	239
	5.92.2 Constructor & Destructor Documentation	239
	5.92.2.1 Trap()	239
	5.92.3 Member Function Documentation	240
	5.92.3.1 get_maximum()	240
	5.92.3.2 has_known_maximum()	240
5.93	Umda Class Reference	241
	5.93.1 Detailed Description	242
5.94	UniformCrossover Class Reference	243
	5.94.1 Detailed Description	243
	5.94.2 Member Function Documentation	243
	5.94.2.1 breed()	243
5.95	WalshExpansion Class Reference	244
	5.95.1 Detailed Description	246
	5.95.2 Member Function Documentation	246
	5.95.2.1 random()	246
5.96	WalshExpansion1 Class Reference	247
	5.96.1 Detailed Description	249
	5.96.2 Member Function Documentation	249
	5.96.2.1 random()	249
5.97	WalshExpansion2 Class Reference	250
	5.97.1 Detailed Description	251
	5.97.2 Member Function Documentation	251
	5.97.2.1 random()	251
	5.97.3 Member Data Documentation	252
	5.97.3.1 _quadratic	252
Index		253

## **Chapter 1**

# Namespace Index

### 1.1 Namespace List

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

nnco	
Top-level HNCO namespace	11
hnco::algorithm	
Algorithms	18
hnco::algorithm::bm_pbil	
Boltzmann machine PBIL	20
hnco::algorithm::hea	
Herding evolutionary algorithm	21
hnco::exception	
Exceptions	21
hnco::function	
Functions to be maximized	22
hnco::neighborhood	
Neighborhoods for local search	23
hnco::random	
Pseudo random numbers	24

2 Namespace Index

## **Chapter 2**

# **Hierarchical Index**

### 2.1 Class Hierarchy

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

Algorithm	30
CompleteSearch	
IterativeAlgorithm	
BmPbil	
GeneticAlgorithm	
$Hea < Moment, Herding > \ \ldots \$	
MuCommaLambdaEa	
MuPlusLambdaEa	
OnePlusLambdaCommaLambdaGa	
PvAlgorithm	
CompactGa	
Mmas	
NpsPbil	
Pbil	
Umda   2     RandomLocalSearch   1	
RandomSearch	
Restart	
SimulatedAnnealing	-
SteepestAscentHillClimbing	
OnePlusOneEa	
BinaryHerding	36
BinaryMoment	38
Crossover	52
BiasedCrossover	
UniformCrossover	
ProgressTracker::Event	
	61
	-
Error	
LastEvaluation	
•	
LocalMaximum	
TargetReached	.32

4 Hierarchical Index

Function	67
DeceptiveJump	53
EqualProducts	56
Factorization	61
FourPeaks	64
FunctionDecorator	73
FunctionController	70
Cache	43
CallCounter	46
OnBudgetFunction	158
ProgressTracker	186
StopOnMaximum	
StopOnTarget	226
FunctionModifier	
AdditiveGaussianNoise	
FunctionMapComposition	
Negation	
PriorNoise	
FunctionPlugin	
Hiff	
Jump	
Labs	
LeadingOnes	
LinearFunction	
MaxSat	
Needle	
NkLandscape	
OneMax	
Plateau	
Qubo	
Ridge	203
SixPeaks	213
SummationCancellation	229
SinusSummationCancellation	211
Trap	238
WalshExpansion	244
WalshExpansion1	247
WalshExpansion2	250
Iterator	102
Hypercubelterator	98
NeighborhoodIterator	151
HammingBallIterator	87
SingleBitFlipIterator	210
Map	121
AffineMap	28
LinearMap	116
MapComposition	122
Permutation	172
Translation	236
Model	131
ModelParameters	132
Neighborhood	148
MultiBitFlip	137
BernoulliProcess	32
HammingBall	85
HammingSphere	89

2.1 Class Hierarchy 5

SingleBitFlip	20
Population	17
TournamentSelection	23
Random	19
SpinHerding	21
SpinMoment	21

6 Hierarchical Index

## **Chapter 3**

# **Class Index**

### 3.1 Class List

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

AdditiveGaussianNoise	
Additive Gaussian Noise	25
AffineMap	
Affine map	28
Algorithm	
Abstract search algorithm	30
BernoulliProcess	
•	32
BiasedCrossover	
Biased crossover	35
BinaryHerding	
g ,	36
BinaryMoment	
<b>,</b>	38
BmPbil	
	39
Cache	
	43
CallCounter	
	46
CompactGa	40
, , ,	48
CompleteSearch	
	51
Crossover	F.C
	52
DeceptiveJump	<b>-</b> 0
	53
EqualProducts	EC
Equal products	56
	59
ProgressTracker::Event	55
	60
Exception	00
·	61

8 Class Index

Factorizat	tion	
	Factorization	61
FourPeak		64
Function	Four Peaks	64
	Function	67
Function	Controller Function controller	70
Function		70
		73
	MapComposition	
FunctionN	Composition of a function and a map	74
		78
FunctionF		. •
	19	79
GeneticA		00
Hamming		82
_		85
_	BallIterator	
		87
Hamming	·	89
	oment, Herding >	03
		92
Hiff		
Hypercub	<b>,</b>	95
		98
IterativeA		
	Iterative search	00
Iterator	Mayatay ayay bit ya staya	00
Jump	Iterator over bit vectors	02
•	Jump	04
Labs		
	Low autocorrelation binary sequences	07
LastEvalu	lation Last evaluation	na
LeadingO		00
	Leading ones	10
LinearFur		
LinearMa	Linear function	13
	p Linear map	16
LocalMax	·	
	Local maximum	18
LongPath	Long path	10
Мар	Long pain	19
	Мар	21
MapComp		
	Map composition	22
Maximum	iReached Maximum reached	24
MaxSat		
	MAX-SAT 1	26

3.1 Class List

Mmas	
Max-min ant system	129
Model Model of a Boltzmann machine	131
ModelParameters	
Parameters of a Boltzmann machine	132
(mu, lambda) EA	134
•	137
MuPlusLambdaEa (mu+lambda) EA	139
Needle	
Needle in a haystack	142
Negation	145
Neighborhood	148
Neighborhood iterator  Neighborhood iterator	151
NkLandscape	151
NK landscape	152
Population-based incremental learning with negative and positive selection	155
OnBudgetFunction  CallCounter with a limited number of evaluations	158
OneMax	
OneMax	160
	163
OnePlusOneEa (1+1) EA	166
Pbil	
Population-based incremental learning	169
	172
Plateau Plateau	174
PointValueException Point-value exception	176
Population	
PriorNoise	177
Prior noise	183
ProgressTracker ProgressTracker	186
PvAlgorithm	
Probability vector algorithm	190
Quadratic unconstrained binary optimization	193
Random Random numbers	195
RandomLocalSearch Random local search	196
RandomSearch	
Random search	199
Restart	201

10 Class Index

Ridge	
Ridge	3
SimulatedAnnealing	
Simulated annealing	6
SingleBitFlip	
One bit neighborhood	8
SingleBitFlipIterator	
Single bit flip neighborhood iterator	0
SinusSummationCancellation	
Summation cancellation with sinus	1
SixPeaks	_
Six Peaks	3
SpinHerding  Usualing with anim variables	_
Herding with spin variables	О
SpinMoment  Moment for spin variables	0
SteepestAscentHillClimbing	J
Steepest ascent hill climbing	'n
StopOnMaximum	Ů
Stop on maximum	2
StopOnTarget	_
Stop on target	6
SummationCancellation	
Summation cancellation	9
TargetReached	
Target reached	2
TournamentSelection	
Population with tournament selection	4
Translation	
Translation	6
Trap	
Trap	8
Umda	
Univariate marginal distribution algorithm	1
UniformCrossover	
Uniform crossover	3
WalshExpansion	ï
Walsh expansion	4
WalshExpansion 1  Walsh expansion of degree 1	_
Walsh expansion of degree 1	1
Walsh expansion of degree 2	n
vvaisii expansion oi degree 2	U

### **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 Hypercubelterator

Hypercube iterator.

· class Iterator

Iterator over bit vectors.

class LinearMap

Linear map.

• class Map

Мар.

• 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

    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.

    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 vectors

```
typedef char bit_t

Bit.
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.
• bit_t bit_flip (bit_t b)
      Flip bit.

    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 by 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 by 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.

    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.

#### 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()

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()

Invert a bit matrix.

#### **Parameters**

М	input matrix
Ν	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 153 of file bit-matrix.cc.

#### 4.1.3.4 bm\_multiply()

Multiply a bit matrix and a bit vector.

The result is y = Mx.

Definition at line 195 of file bit-matrix.cc.

#### 4.1.3.5 bm\_solve()

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

Solve a linear system.

Solve the linear equation Ax = b.

#### **Parameters**

Α	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 103 of file bit-matrix.cc.

#### 4.1.3.6 bm\_solve\_upper\_triangular()

Solve a linear system in upper triangular form.

Solve the linear equation Ax = b.

#### **Parameters**

Α	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 134 of file bit-matrix.cc.

#### 4.1.3.7 sbm\_multiply()

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 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)</li>
 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 HammingBallIterator

Hamming ball neighborhood iterator.

• class HammingSphere

Hamming sphere.

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.

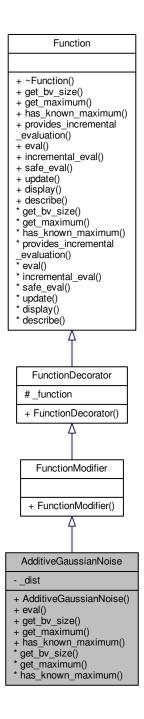
# Chapter 5

# **Class Documentation**

5.1 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

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:

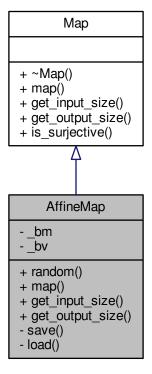
- · lib/hnco/functions/decorators/function-modifier.hh
- · lib/hnco/functions/decorators/function-modifier.cc

# 5.2 AffineMap Class Reference

Affine map.

#include <hnco/map.hh>

Inheritance diagram for AffineMap:



#### **Public Member Functions**

```
• void random (int n, int m)
```

Random instance.

void map (const bit\_vector\_t &input, bit\_vector\_t &output)

Мар.

• size\_t get\_input\_size ()

Get input size.

• size\_t get\_output\_size ()

Get output size.

#### **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  $\mathbb{Z}_2^m$  to  $\mathbb{Z}_2^n$  is defined by f(x) = Ax + b, where A is an n x m bit matrix and b is an n-dimensional bit vector.

#### Warning

The class does not reimplement the member function is\_surjective hence a linear map is always considered not surjective.

Definition at line 258 of file map.hh.

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

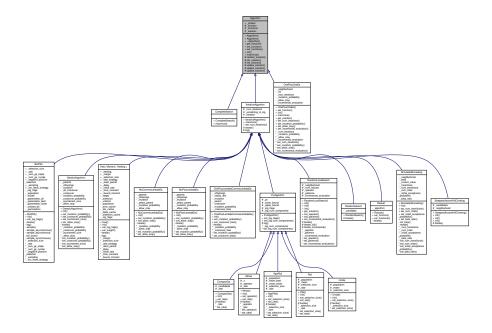
- · lib/hnco/map.hh
- lib/hnco/map.cc

# 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 const point\_value\_t & get\_solution ()

Solution

virtual void set\_function (function::Function \*function)

Set function.

virtual void set\_functions (const std::vector< function::Function \*> functions)

Set functions.

· virtual void init ()

Initialization.

• virtual void maximize ()=0

Maximize.

# **Public Attributes**

• std::ostream & \_stream = std::cout 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.

# 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:

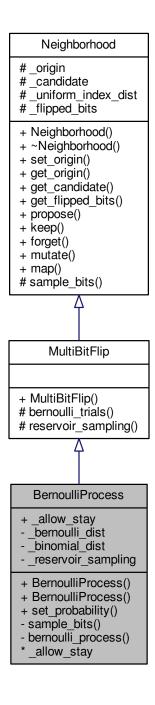
- · lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

# 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.

#### **Public Attributes**

#### **Parameters**

```
    bool <u>_allow_stay</u> = false

        Allow stay.
```

#### **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 <u>\_reservoir\_sampling</u> = false

Reservoir sampling.

#### **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 246 of file neighborhood.hh.

# 5.4.2.2 BernoulliProcess() [2/2]

```
BernoulliProcess (  \mbox{int } n, \\ \mbox{double } p \mbox{ ) } \mbox{ [inline]}
```

Constructor.

#### **Parameters**

n	Size of bit vectors	
р	Bernoulli probability	

Definition at line 256 of file neighborhood.hh.

# 5.4.3 Member Function Documentation

# 5.4.3.1 set\_probability()

```
void set_probability ( \label{eq:condition} \mbox{double } p \mbox{ ) } \mbox{ [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 267 of file neighborhood.hh.

#### 5.4.4 Member Data Documentation

#### 5.4.4.1 \_allow\_stay

bool \_allow\_stay = false

Allow stay.

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

Definition at line 283 of file neighborhood.hh.

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

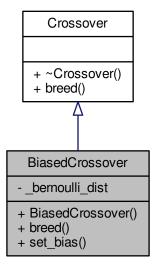
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 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()

#### Breed.

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

#### **Parameters**

parent1	First parent
parent2	Second parent
offspring	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.

• double error (const BinaryMoment &target)

Compute the error.

double delta (const BinaryMoment &target)

Compute the norm of delta.

void sample (const BinaryMoment &target, bit\_vector\_t &x)

Sample a bit vector.

#### **Public Attributes**

#### **Parameters**

• bool <u>\_randomize\_bit\_order</u> = false

Randomize bit order.

int \_dynamics = DYNAMICS\_MINIMIZE\_NORM

Dynamics.

• double \_weight = 1

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

 $\bullet \quad \mathsf{std}{::} \mathsf{uniform\_int\_distribution}{<} \; \mathsf{int} > \underline{\mathsf{-choose\_bit}} \\$ 

Choose bit.

• int \_time

Time.

# 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

DYNAMICS_MINIMIZE_NORM	Dynamics defined as minimization of a norm.
DYNAMICS_MAXIMIZE_INNER_PRODUCT	Dynamics defined as maximization of an inner product.

Definition at line 69 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:

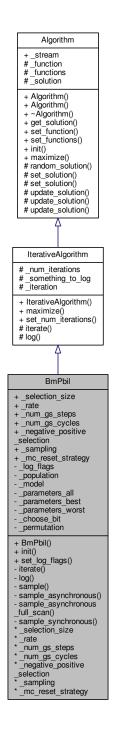
- lib/hnco/algorithms/hea/moment-binary.hh
- lib/hnco/algorithms/hea/moment-binary.cc

# 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 { 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.

• void set\_log\_flags (const log\_flags\_t &lf)

Set log flags.

#### **Public Attributes**

#### **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.

#### **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.

#### **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.

5.9 Cache Class Reference 43

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 _num_gs_steps 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.

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

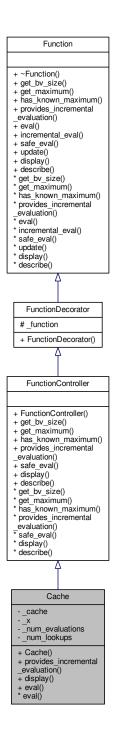
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

# 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.

· void display (std::ostream &stream)

5.9 Cache Class Reference 45

Display.

#### **Evaluation**

double eval (const bit\_vector\_t &)
 Evaluate a bit vector.

#### **Private Attributes**

```
    std::unordered_map< std::vector< bool >, double > _cache
    Database of past evaluations.
```

•  $std::vector < bool > \underline{x}$ 

STL bit vector.

• 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 344 of file function-controller.hh.

#### 5.9.2 Constructor & Destructor Documentation

# 5.9.2.1 Cache()

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

Constructor.

**Parameters** 

function Decorated function

Definition	at line	363	of file	function.	-controller hh	١.

#### 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 372 of file function-controller.hh.

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

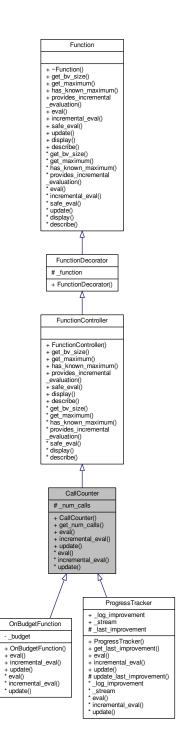
- lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 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:

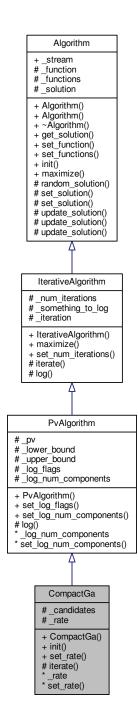
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 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**

# **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:

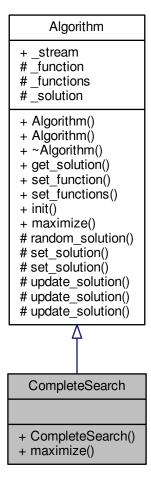
- · lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

# 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:

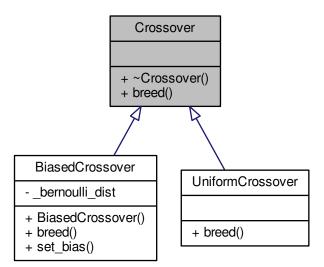
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

# 5.13 Crossover Class Reference

#### Crossover.

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

Inheritance diagram for Crossover:



# **Public Member Functions**

- virtual  $\sim$ Crossover ()
- 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()

#### Breed.

The offspring is the crossover of two parents.

#### **Parameters**

parent1	First parent
parent2	Second parent
offspring	Offspring

Implemented in BiasedCrossover, and UniformCrossover.

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

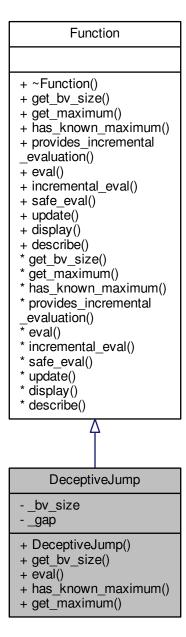
• lib/hnco/algorithms/ea/crossover.hh

# 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.

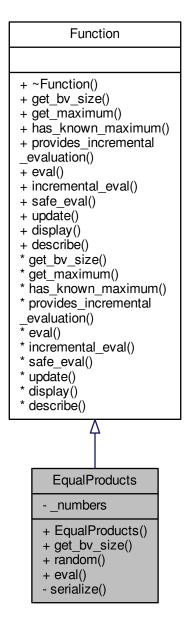
Returns

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:
<ul> <li>lib/hnco/functions/jump.hh</li> <li>lib/hnco/functions/jump.cc</li> </ul>
5.15 EqualProducts Class Reference
Equal products.
<pre>#include <hnco equal-products.hh="" functions=""></hnco></pre>

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**

n	Size of bit vector
upper_bound	Upper bound of numbers

5.16 Error Class Reference 59

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

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

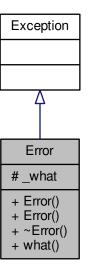
- · lib/hnco/functions/equal-products.hh
- lib/hnco/functions/equal-products.cc

# 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:

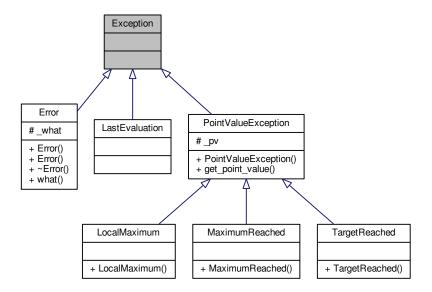
· lib/hnco/functions/decorators/function-controller.hh

# 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:

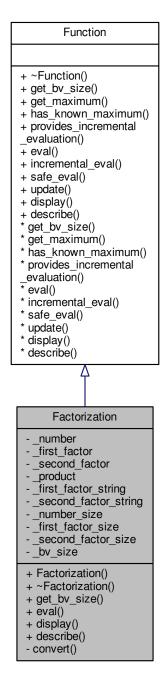
• lib/hnco/exception.hh

# 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** 

path 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:

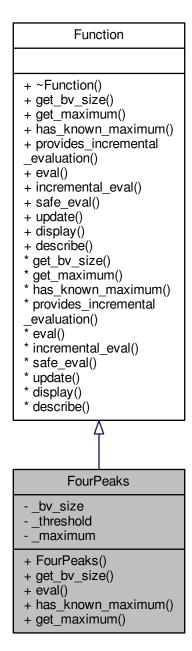
- · lib/hnco/functions/factorization.hh
- · lib/hnco/functions/factorization.cc

# 5.20 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.

# 5.20.1 Detailed Description

Four Peaks.

It is defined by

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

where:

- head(x, 1) is the length of the longest prefix of x made of ones;
- tail(x, 0) is the length of the longest suffix of x made of zeros;
- R(x) is the reward;
- R(x) = n if (head(x, 1) > t and 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(1111110) = 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.

### 5.20.2 Member Function Documentation

# 5.20.2.1 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.

### 5.20.2.2 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:

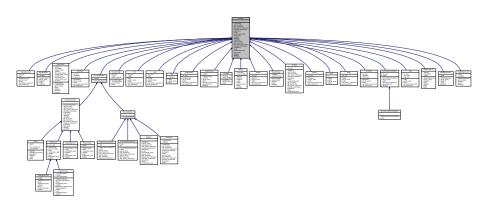
- · lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

# 5.21 Function Class Reference

### Function.

#include <hnco/functions/function.hh>

Inheritance diagram for Function:



### **Public Member Functions**

• virtual  $\sim$ 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.

### **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.

# 5.21.1 Detailed Description

### Function.

Definition at line 35 of file function.hh.

# 5.21.2 Member Function Documentation

```
5.21.2.1 get_maximum()
```

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

Get the global maximum.

### **Exceptions**

Error

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

Definition at line 51 of file function.hh.

### 5.21.2.2 incremental\_eval()

Incremental evaluation.

### **Exceptions**

Error

Reimplemented in OnBudgetFunction, ProgressTracker, CallCounter, StopOnTarget, StopOnMaximum, Negation, and OneMax.

Definition at line 75 of file function.hh.

# 5.21.2.3 provides\_incremental\_evaluation()

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

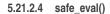
Check whether the function provides incremental evaluation.

Returns

false

Reimplemented in Cache, Negation, PriorNoise, OneMax, and FunctionController.

Definition at line 59 of file function.hh.



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 85 of file function.hh.

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

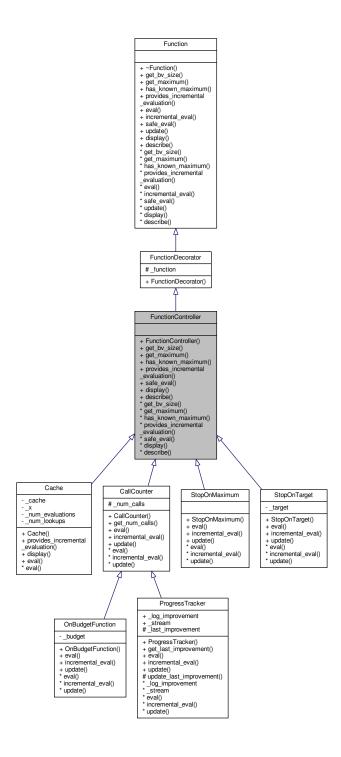
• lib/hnco/functions/function.hh

# 5.22 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**

### 5.22.1 Detailed Description

Function controller.

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

### 5.22.2 Member Function Documentation

```
5.22.2.1 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:

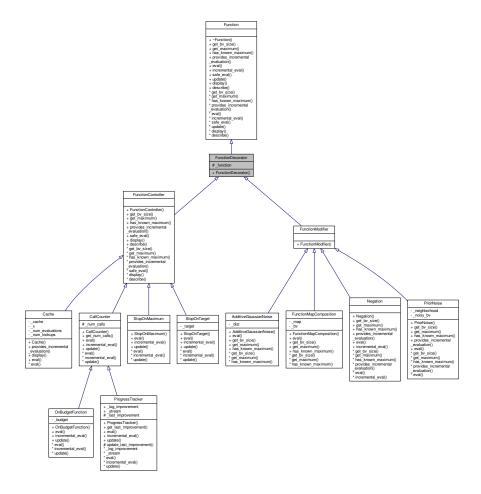
· lib/hnco/functions/decorators/function-controller.hh

# 5.23 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.

74	Class Documentation
5.23.1 Detailed Description	
Function decorator.	

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

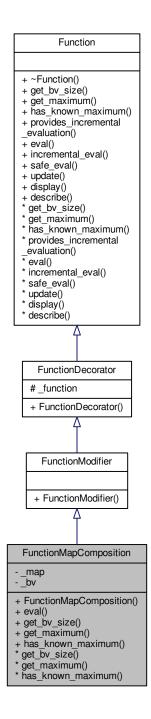
The documentation for this class was generated from the following file:

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

# 5.24 FunctionMapComposition Class Reference

Composition of a function and a map.

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 * _mapMap.bit_vector_t _bv
```

### **Additional Inherited Members**

# 5.24.1 Detailed Description

Composition of a function and a map.

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

### 5.24.2 Constructor & Destructor Documentation

### 5.24.2.1 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.

### 5.24.3 Member Function Documentation

# 5.24.3.1 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.

### 5.24.3.2 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:

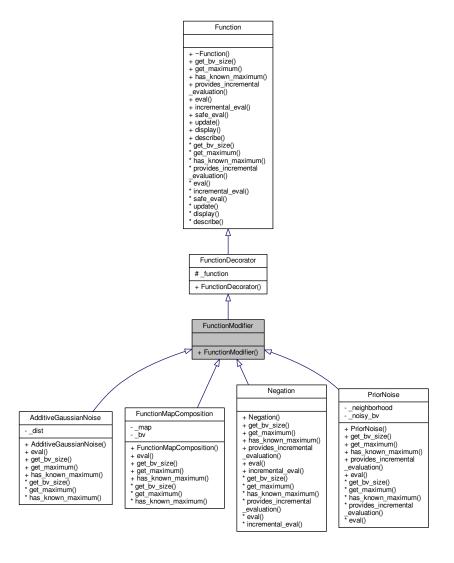
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

# 5.25 FunctionModifier Class Reference

Function modifier.

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

Inheritance diagram for FunctionModifier:



# **Public Member Functions**

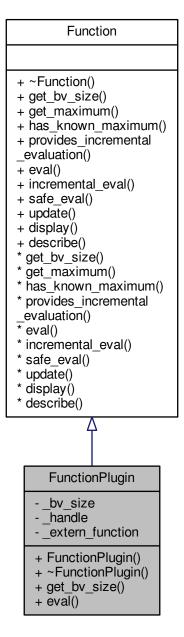
• FunctionModifier (Function \*function)

Constructor.

Additional Inherited Members
5.25.1 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
5.26 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_sizeBit vector size.void * _handle
```

\_

Handle returned by dlopen.

extern\_function\_t \_extern\_function
 Extern function.

# 5.26.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

### 5.26.2 Constructor & Destructor Documentation

# 5.26.2.1 FunctionPlugin()

### Constructor.

### **Parameters**

bv_size	Size of bit vectors
path	Path to a shared library
name	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:

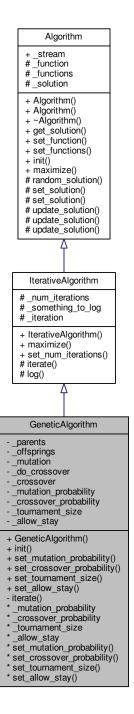
- · lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

# 5.27 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**

# 5.27.1 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.

# 5.27.2 Constructor & Destructor Documentation

# 5.27.2.1 GeneticAlgorithm()

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

Constructor.

### **Parameters**

n	Size of bit vectors
mu	Population size

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

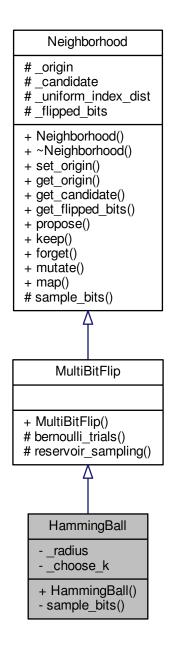
# 5.27.3 Member Function Documentation

# 5.27.3.1 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
5.28 HammingBall Class Reference
Hamming ball.
#include <hnco hh="" neighborhood="" neighborhoods=""></hnco>

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**

### 5.28.1 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 295 of file neighborhood.hh.

### 5.28.2 Constructor & Destructor Documentation

### 5.28.2.1 HammingBall()

```
\label{eq:balance} \begin{array}{ll} \mbox{HammingBall (} \\ & \mbox{int } n, \\ & \mbox{int } r \mbox{ ) } \mbox{ [inline]} \end{array}
```

Constructor.

### **Parameters**

n	Size of bit vectors
r	Radius of the ball

Definition at line 314 of file neighborhood.hh.

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

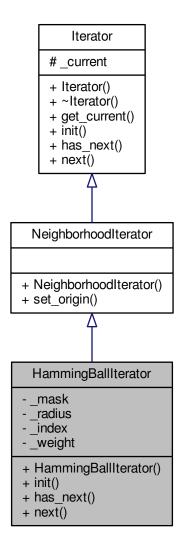
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 5.29 HammingBallIterator Class Reference

Hamming ball neighborhood iterator.

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

Inheritance diagram for HammingBallIterator:



# **Public Member Functions**

• HammingBallIterator (int n, int r)

Constructor.

• void init ()

Initialization.

• bool has\_next ()

Has next bit vector.

• void 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**

# 5.29.1 Detailed Description

Hamming ball neighborhood iterator.

Definition at line 86 of file neighborhood-iterator.hh.

# 5.29.2 Constructor & Destructor Documentation

### 5.29.2.1 HammingBallIterator()

Constructor.

### **Parameters**

n	Size of bit vectors
r	Radius of Hamming Ball

Definition at line 108 of file neighborhood-iterator.hh.

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

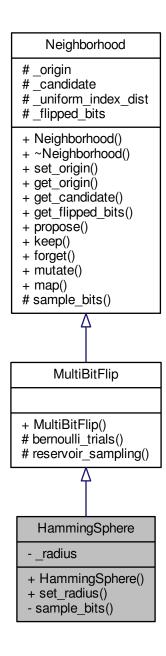
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

# 5.30 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 332 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 348 of file neighborhood.hh.

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

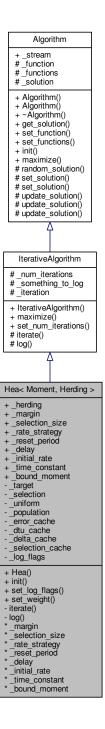
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 5.31 Hea < Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



### **Public Types**

```
    enum {
        LOG_ERROR, LOG_DTU, LOG_DELTA, LOG_SELECTION,
        LAST_LOG }
    enum { RATE_CONSTANT, RATE_EXPONENTIAL, RATE_INVERSE }
    typedef std::bitset < LAST_LOG > log_flags_t
```

### **Public Member Functions**

```
• Hea (int n, int population_size)
```

Type for log flags.

Constructor.

• void init ()

Initialization.

• void set\_log\_flags (const log\_flags\_t &lf)

Set log flags.

• void set\_weight (double weight)

Set weight.

# **Public Attributes**

Herding \_herding
 Herding.

# **Parameters**

```
    double _margin
        Moment margin.
    int _selection_size = 1
        Selection size (number of selected individuals in the population)
    int _rate_strategy = RATE_CONSTANT
        Rate strategy.
    int _reset_period = 0
        Reset period (<= 0 means no reset)</li>
    int _delay = 10000
        Delay.
    double _initial_rate = 1e-4
        Initial value of the learning rate.
```

Time constant.bool \_bound\_moment = false

• double \_time\_constant = 1000

Bound moment after update.

#### **Private Member Functions**

```
void iterate ()
Single iteration.
void log ()
Log.
```

#### **Private Attributes**

Moment <u>\_target</u>

Moment.

• Moment \_selection

Moment of selected individuals.

• Moment <u>uniform</u>

Uniform moment.

• algorithm::Population \_population

Population.

• 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.

#### **Additional Inherited Members**

# 5.31.1 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.

#### 5.31.2 Member Enumeration Documentation

#### 5.31.2.1 anonymous enum

anonymous enum

5.32 Hiff Class Reference 95

#### 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 54 of file hea.hh.

# 5.31.2.2 anonymous enum

anonymous enum

#### Enumerator

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

Definition at line 202 of file hea.hh.

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

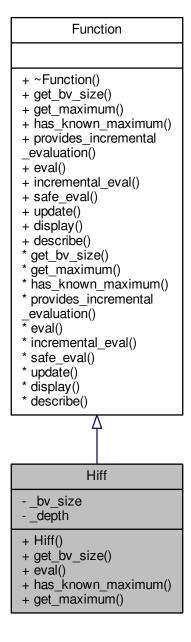
• lib/hnco/algorithms/hea/hea.hh

# 5.32 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.

5.32 Hiff Class Reference 97

```
• 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.

# 5.32.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 165 of file theory.hh.

# 5.32.2 Member Function Documentation

```
5.32.2.1 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.

#### 5.32.2.2 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:

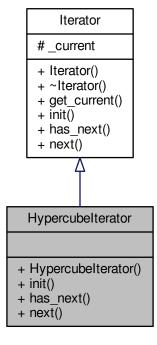
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

# 5.33 Hypercubelterator Class Reference

Hypercube iterator.

#include <hnco/iterator.hh>

Inheritance diagram for Hypercubelterator:



# **Public Member Functions**

• Hypercubelterator (int n)

Constructor.

· void init ()

Initialization.

• bool has next ()

Has next bit vector.

• void next ()

Next bit vector.

# **Additional Inherited Members**

# 5.33.1 Detailed Description

Hypercube iterator.

Definition at line 71 of file iterator.hh.

### 5.33.2 Member Function Documentation

```
5.33.2.1 next()
```

```
void next ( ) [virtual]
```

Next bit vector.

Implemented as a simple binary adder.

Implements Iterator.

Definition at line 28 of file iterator.cc.

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

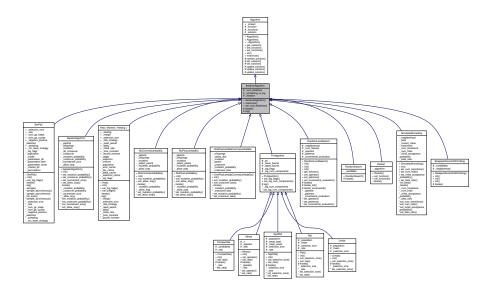
- · lib/hnco/iterator.hh
- lib/hnco/iterator.cc

# 5.34 Iterative Algorithm Class Reference

#### Iterative search.

#include <hnco/algorithms/algorithm.hh>

# Inheritance diagram for IterativeAlgorithm:



# **Public Member Functions**

• IterativeAlgorithm (int n)

Constructor.

• void maximize ()

Maximize.

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 \_num\_iterations = 0

Number of iterations.

- bool \_something\_to\_log Something to log.
- int \_iteration

Current iteration.

**Additional Inherited Members** 

# 5.34.1 Detailed Description

Iterative search.

Definition at line 115 of file algorithm.hh.

#### 5.34.2 Constructor & Destructor Documentation

#### 5.34.2.1 IterativeAlgorithm()

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

Constructor.

**Parameters** 

```
n Size of bit vectors
```

Definition at line 139 of file algorithm.hh.

# 5.34.3 Member Function Documentation

# 5.34.3.1 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.

# 5.34.3.2 set\_num\_iterations()

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

Set the number of iterations.

**Parameters** 

x Number of iterations

 $x \le 0$  means indefinite

Definition at line 159 of file algorithm.hh.

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

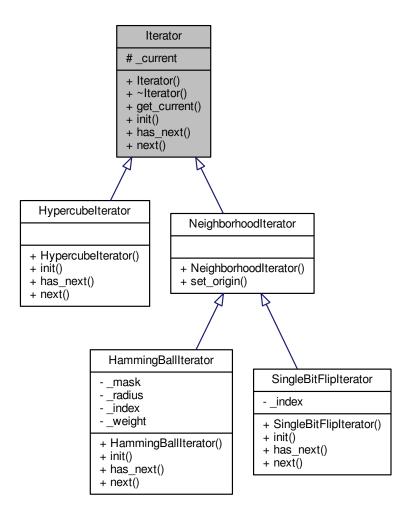
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

# 5.35 Iterator Class Reference

Iterator over bit vectors.

#include <hnco/iterator.hh>

Inheritance diagram for Iterator:



# **Public Member Functions**

• Iterator (int n)

Constructor.

virtual ∼lterator ()

Destructor.

• virtual const bit\_vector\_t & get\_current ()

Current bit vector.

• virtual void init ()=0

Initialization.

• virtual bool has\_next ()=0

Has next bit vector.

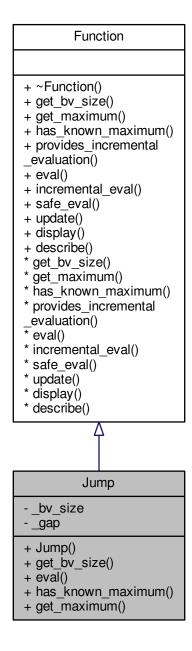
virtual void next ()=0

Next bit vector.

Protected Attributes		
• bit_vector_t _current  Current bit vector.		
5.35.1 Detailed Description		
Iterator over bit vectors.		
Definition at line 40 of file iterator.hh.		
The documentation for this class was generated from the following file:		
• lib/hnco/iterator.hh		
5.36 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.

# 5.36.1 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.

# 5.36.2 Member Function Documentation

```
5.36.2.1 get_maximum()
```

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

Get the global maximum.

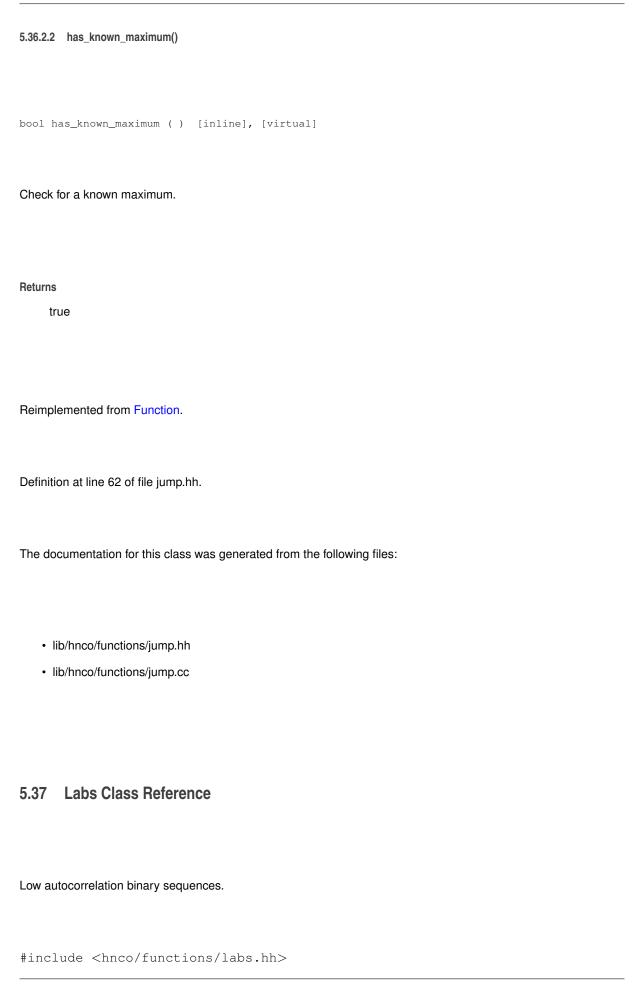
Returns

\_bv\_size

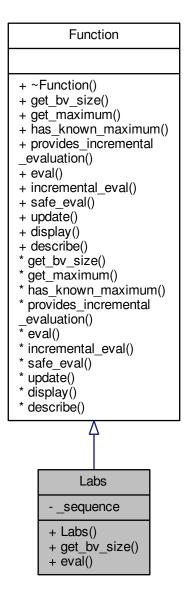
Reimplemented from Function.

Definition at line 66 of file jump.hh.

5.37 Labs Class Reference 107



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.

#### 5.37.1 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:

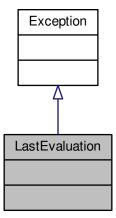
- · lib/hnco/functions/labs.hh
- lib/hnco/functions/labs.cc

# 5.38 LastEvaluation Class Reference

Last evaluation.

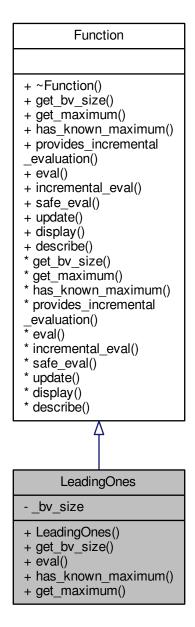
#include <hnco/exception.hh>

Inheritance diagram for LastEvaluation:



5.38.1 D	Detailed Description
Last evalu	nation.
Definition	at line 79 of file exception.hh.
The docur	mentation for this class was generated from the following file:
• lib/h	nnco/exception.hh
5.39 L	eadingOnes Class Reference
Leading o	nes.
#includ	de <hnco functions="" theory.hh=""></hnco>

Inheritance diagram for LeadingOnes:



### **Public Member Functions**

• LeadingOnes (int by 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.
```

#### 5.39.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 93 of file theory.hh.

#### 5.39.2 Member Function Documentation

```
5.39.2.1 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.

```
5.39.2.2 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

# 5.40 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:

# Function + ~Function() + get\_bv\_size() + get\_maximum() + has\_known\_maximum() + provides\_incremental \_evaluation() + eval() + incremental\_eval() + safe\_eval() + update() + display() + describe() \* get\_bv\_size() \* get\_maximum() \* has\_known\_maximum() \* provides\_incremental \_evaluation() \* eval() \* incremental\_eval() \* safe\_eval() \* update() \* display() \* describe() LinearFunction - \_weights + LinearFunction() + random() + get\_bv\_size() + eval() + has\_known\_maximum() + get maximum() - serialize()

#### **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

# 5.40.1 Detailed Description

Linear function.

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

#### 5.40.2 Member Function Documentation

```
5.40.2.1 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.

# 5.40.2.2 random()

```
void random ( \quad \text{int } n \ )
```

Random instance.

#### **Parameters**

n Size of bit vectors

Definition at line 34 of file linear-function.cc.

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

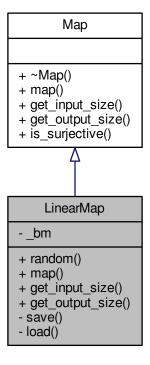
- lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

# 5.41 LinearMap Class Reference

Linear map.

#include <hnco/map.hh>

Inheritance diagram for LinearMap:



#### **Public Member Functions**

```
• void random (int n, int m)
```

Random instance.

void map (const bit\_vector\_t &input, bit\_vector\_t &output)

Мар.

• size\_t get\_input\_size ()

Get input size.

• size\_t get\_output\_size ()

Get output size.

# **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 \_bmBit matrix.

### **Friends**

· class boost::serialization::access

# 5.41.1 Detailed Description

Linear map.

A linear map f from  $\mathbb{Z}_2^m$  to  $\mathbb{Z}_2^n$  is defined by f(x)=Ax, where A is an m x n bit matrix.

#### Warning

The class does not reimplement the member function is\_surjective hence a linear map is always considered not surjective.

Definition at line 197 of file map.hh.

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

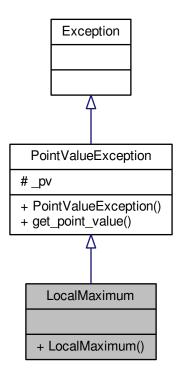
- · lib/hnco/map.hh
- lib/hnco/map.cc

# 5.42 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**

# 5.42.1 Detailed Description

Local maximum.

Definition at line 70 of file exception.hh.

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

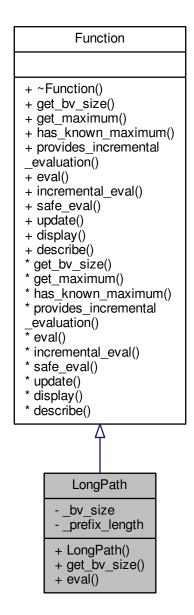
• lib/hnco/exception.hh

# 5.43 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.
```

#### 5.43.1 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:

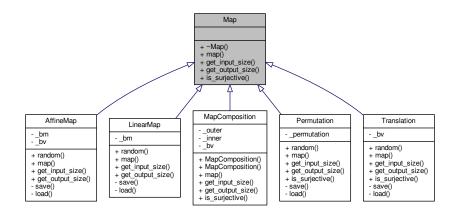
- · lib/hnco/functions/long-path.hh
- lib/hnco/functions/long-path.cc

# 5.44 Map Class Reference

#### Мар.

#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

Мар.

• 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.

#### 5.44.1 Detailed Description

#### Map.

Definition at line 39 of file map.hh.

### 5.44.2 Member Function Documentation

#### 5.44.2.1 is\_surjective()

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

Check for surjective map.

Returns

false

Reimplemented in MapComposition, Permutation, and Translation.

Definition at line 59 of file map.hh.

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

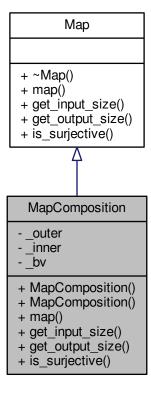
· lib/hnco/map.hh

# 5.45 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)

Man.

• 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.

# 5.45.1 Detailed Description

Map composition.

The resulting composition f is defined for all bit vector x by f(x) = outer(inner(x)).

Definition at line 326 of file map.hh.

#### 5.45.2 Constructor & Destructor Documentation

#### 5.45.2.1 MapComposition()

Constructor.

#### **Parameters**

outer	outer map
inner	inner map



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

Definition at line 350 of file map.hh.

# 5.45.3 Member Function Documentation

# 5.45.3.1 is\_surjective()

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

Check for surjective map.

# Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 374 of file map.hh.

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

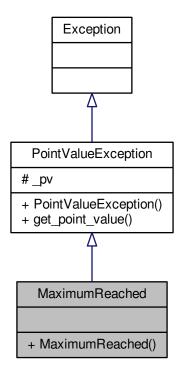
· lib/hnco/map.hh

# 5.46 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** 

# 5.46.1 Detailed Description

Maximum reached.

Definition at line 52 of file exception.hh.

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

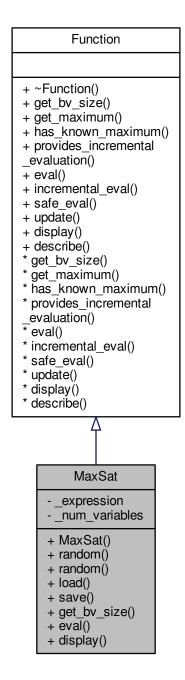
lib/hnco/exception.hh

# 5.47 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.

# 5.47.1 Detailed Description

MAX-SAT.

Reference:

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

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

#### 5.47.2 Member Function Documentation

Load an instance.

# **Exceptions**

```
Error
```

Definition at line 134 of file max-sat.cc.

Random instance.

# **Parameters**

n	Size of bit vectors
k	Number of literals per clause
С	Number of clauses

Definition at line 39 of file max-sat.cc.

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**

solution	Solution
k	Number of literals per clause
С	Number of clauses

Definition at line 67 of file max-sat.cc.

5.48 Mmas Class Reference 129

5.47.3	Member	Data	<b>Documentation</b>
U.T1.U	IVICIIIDCI	Dutu	Documentation

5.47.3.1 \_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:

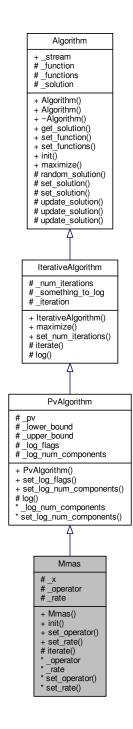
- · lib/hnco/functions/max-sat.hh
- lib/hnco/functions/max-sat.cc

# 5.48 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.

5.49 Model Class Reference 131

#### **Setters**

void set\_operator (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)> \_operator = std::greater\_equal<double>()

  Binary operator for comparing evaluations.
- double <u>\_rate</u> = 1e-3 <u>Learning rate</u>.

#### **Additional Inherited Members**

# 5.48.1 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

# 5.49 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\_I1 ()

I1 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.

# 5.49.1 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

# 5.50 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 I1 ()

I1 norm of the parameters

#### **Private Attributes**

```
std::vector< std::vector< double >> _weight
```

Weights.

std::vector< double > \_bias

Bias.

# **Friends**

· class Model

# 5.50.1 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:

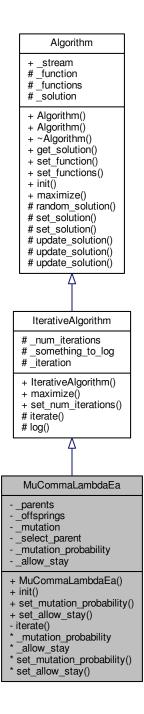
- lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

# 5.51 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.

 $\bullet \quad \mathsf{std} :: \mathsf{uniform\_int\_distribution} < \mathsf{int} > \_\mathsf{select\_parent}$ 

Select parent.

### **Parameters**

· double mutation probability

Mutation probability.

bool <u>\_allow\_stay</u> = false

Allow stay.

#### **Additional Inherited Members**

# 5.51.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

# 5.51.2 Constructor & Destructor Documentation

# 5.51.2.1 MuCommaLambdaEa()

```
\begin{tabular}{lll} MuCommaLambdaEa & ( & & int $n$, & & \\ & & int $mu$, & \\ & & int $lambda$ ) & [inline] \end{tabular}
```

Constructor.

#### **Parameters**

n	Size of bit vectors
mu	Parent population size
lambda	Offspring population size

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

# 5.51.3 Member Function Documentation

# 5.51.3.1 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:

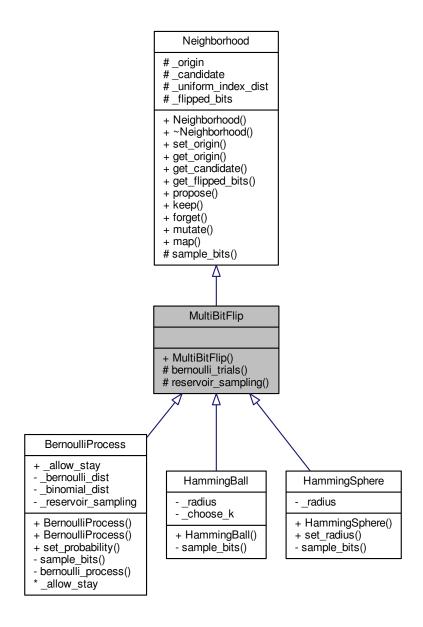
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

# 5.52 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**

# 5.52.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

# 5.52.2 Constructor & Destructor Documentation

# 5.52.2.1 MultiBitFlip()

```
\label{eq:multiBitFlip} \mbox{MultiBitFlip (} & \mbox{int } n \mbox{ ) } \mbox{[inline]}
```

Constructor.

### **Parameters**

```
n | Size of bit vectors
```

Definition at line 206 of file neighborhood.hh.

# 5.52.3 Member Function Documentation

# 5.52.3.1 bernoulli\_trials()

```
\begin{tabular}{ll} \beg
```

Sample a given number of bits using Bernoulli trials.

### **Parameters**

*k* Number of bits to sample

Definition at line 34 of file neighborhood.cc.

# 5.52.3.2 reservoir\_sampling()

```
\begin{tabular}{ll} \beg
```

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:

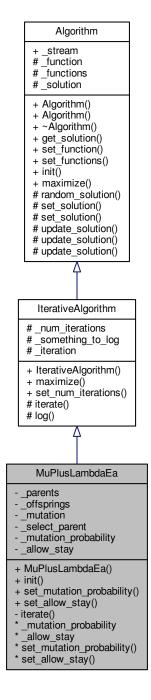
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 5.53 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.

 $\bullet \quad \mathsf{std::} \mathsf{uniform\_int\_distribution} < \mathsf{int} > \_\mathsf{select\_parent}$ 

Select parent.

# **Parameters**

```
    double _mutation_probability
Mutation probability.
```

```
• bool <u>_allow_stay</u> = false

Allow stay.
```

#### **Additional Inherited Members**

# 5.53.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

# 5.53.2 Constructor & Destructor Documentation

# 5.53.2.1 MuPlusLambdaEa()

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

Constructor.

#### **Parameters**

n	Size of bit vectors
mu	Parent population size
lambda	Offspring population size

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

# 5.53.3 Member Function Documentation

# 5.53.3.1 set\_allow\_stay()

```
void set_allow_stay ( bool \ x \ ) \quad [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:

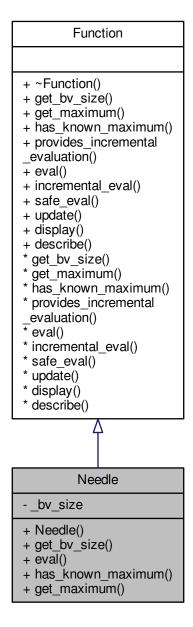
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

# 5.54 Needle Class Reference

Needle in a haystack.

#include <hnco/functions/theory.hh>

Inheritance diagram for Needle:



# **Public Member Functions**

- Needle (int by 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.
```

# 5.54.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 129 of file theory.hh.

### 5.54.2 Member Function Documentation

```
5.54.2.1 get_maximum()

double get_maximum ( ) [inline], [virtual]

Get the global maximum.

Returns
```

Reimplemented from Function.

Definition at line 153 of file theory.hh.

```
5.54.2.2 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:

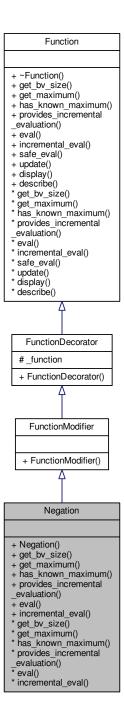
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

# 5.55 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**

# 5.55.1 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.

#### 5.55.2 Member Function Documentation

```
5.55.2.1 get_maximum()
```

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

Get the global maximum.

Exceptions  Error
Reimplemented from Function.
Definition at line 76 of file function-modifier.hh.
5.55.2.2 has_known_maximum()
<pre>bool has_known_maximum ( ) [inline], [virtual]</pre>
Check for a known maximum.
Returns false
Reimplemented from Function.
Definition at line 80 of file function-modifier.hh.
5.55.2.3 provides_incremental_evaluation()
<pre>bool provides_incremental_evaluation ( ) [inline], [virtual]</pre>
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:

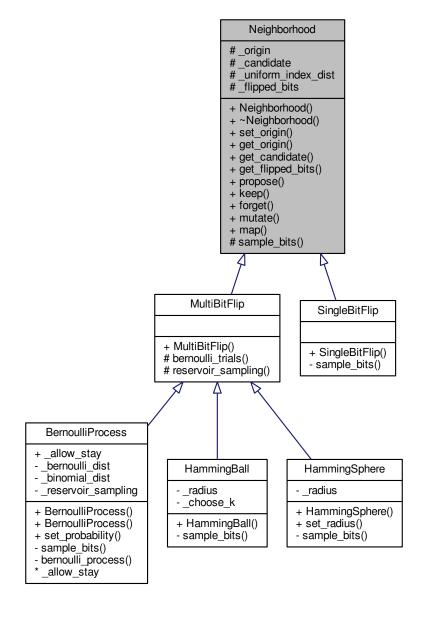
- lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

# 5.56 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)

Мар.

#### **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

 $\bullet \quad \mathsf{std} :: \mathsf{uniform\_int\_distribution} < \mathsf{int} > \underline{\quad \mathsf{uniform\_index\_dist}}$ 

Uniform index distribution.

sparse\_bit\_vector\_t \_flipped\_bits

Flipped bits.

# 5.56.1 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.

# 5.56.2 Constructor & Destructor Documentation

# 5.56.2.1 Neighborhood()

```
Neighborhood (  \qquad \qquad \text{int } n \text{ ) } \quad [\text{inline}]
```

Constructor.

#### **Parameters**

```
n Size of bit vectors
```

Definition at line 86 of file neighborhood.hh.

# 5.56.3 Member Function Documentation

#### 5.56.3.1 map()

Мар.

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

#### **Parameters**

input	Input bit vector
output	Output bit vector

Definition at line 148 of file neighborhood.hh.

# 5.56.3.2 mutate()

Mutate.

In-place mutation of the bit vector.

#### **Parameters**

bv Bit vector to mutate

Definition at line 134 of file neighborhood.hh.

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

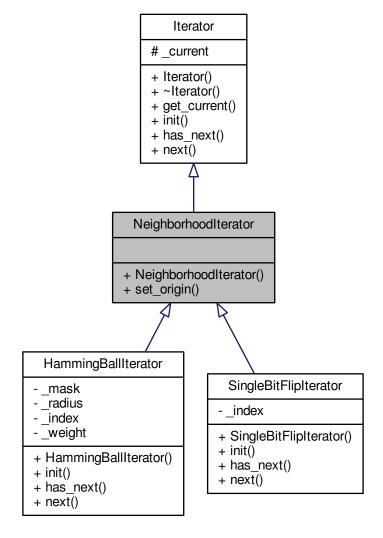
• lib/hnco/neighborhoods/neighborhood.hh

# 5.57 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**

# 5.57.1 Detailed Description

Neighborhood iterator.

Definition at line 38 of file neighborhood-iterator.hh.

# 5.57.2 Constructor & Destructor Documentation

# 5.57.2.1 NeighborhoodIterator()

```
\label{eq:neighborhoodIterator} \mbox{NeighborhoodIterator (} \\ \mbox{int } n \mbox{ ) } \mbox{[inline]}
```

Constructor.

### **Parameters**

```
n Size of bit vectors
```

Definition at line 49 of file neighborhood-iterator.hh.

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

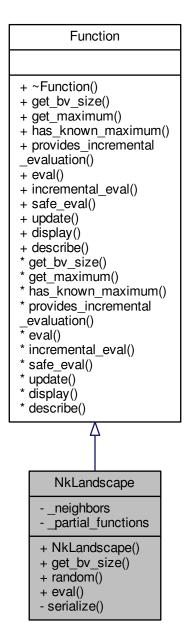
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

# 5.58 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

# 5.58.1 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.

# 5.58.2 Member Function Documentation

#### 5.58.2.1 random()

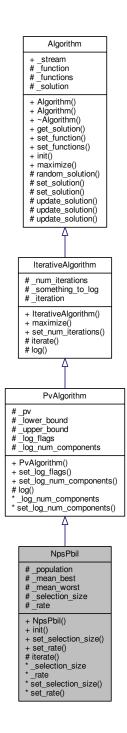
#### Random instance.

### **Parameters**

n	Size of bit vector
k	Number of neighbors of each bit
stddev	Standard deviation of the values of the partial functions

Definition at line 33 of file nk-landscape.cc.
The documentation for this class was generated from the following files:
<ul> <li>lib/hnco/functions/nk-landscape.hh</li> <li>lib/hnco/functions/nk-landscape.cc</li> </ul>
5.59 NpsPbil Class Reference
Population-based incremental learning with negative and positive selection.
<pre>#include <hnco algorithms="" nps-pbil.hh="" pv=""></hnco></pre>

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 <u>\_rate</u> = 1e-3

Learning rate.

# **Additional Inherited Members**

# 5.59.1 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

# 5.60 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**

# 5.60.1 Detailed Description

CallCounter with a limited number of evaluations.

Definition at line 299 of file function-controller.hh.

# 5.60.2 Member Function Documentation

```
5.60.2.1 eval()
```

Evaluate a bit vector.

**Exceptions** 

LastEvaluation

Reimplemented from CallCounter.

Definition at line 121 of file function-controller.cc.

### 5.60.2.2 incremental\_eval()

Incremental evaluation.

**Exceptions** 

LastEvaluation

Reimplemented from CallCounter.

Definition at line 132 of file function-controller.cc.

#### 5.60.2.3 update()

Update after a safe evaluation.

**Exceptions** 

LastEvaluation

Reimplemented from CallCounter.

Definition at line 143 of file function-controller.cc.

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

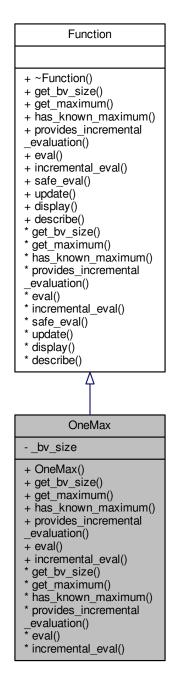
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 5.61 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.

# 5.61.1 Detailed Description

# OneMax.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 36 of file theory.hh.

# 5.61.2 Member Function Documentation

```
5.61.2.1 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.

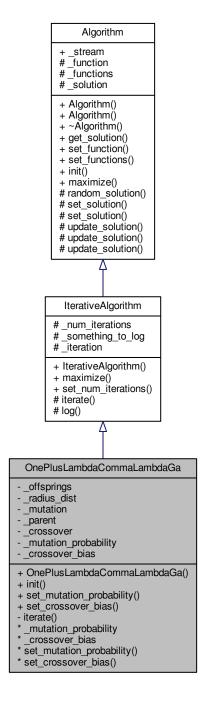
# 5.61.2.2 has\_known\_maximum() bool has\_known\_maximum ( ) [inline], [virtual] Check for a known maximum. Returns true Reimplemented from Function. Definition at line 61 of file theory.hh. 5.61.2.3 provides\_incremental\_evaluation() bool provides\_incremental\_evaluation ( ) [inline], [virtual]Check 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

# 5.62 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**

## 5.62.1 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.

#### 5.62.2 Constructor & Destructor Documentation

## 5.62.2.1 OnePlusLambdaCommaLambdaGa()

```
OnePlusLambdaCommaLambdaGa (  \mbox{int } n, \\ \mbox{int } lambda \mbox{) [inline]}
```

#### Constructor.

By default, \_mutation\_probability is set to lambda / n and \_crossover\_bias to 1 / lambda.

## **Parameters**

n	Size of bit vectors
lambda	Offspring population size

Definition at line 92 of file one-plus-lambda-comma-lambda-ga.hh.

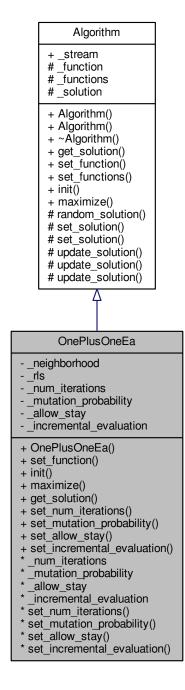
The documentation for this class was generated from the following files:

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

# 5.63 OnePlusOneEa Class Reference

(1+1) EA.

Inheritance diagram for OnePlusOneEa:



## **Public Member Functions**

OnePlusOneEa (int n)

 ${\it Constructor.}$ 

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 <u>\_allow\_stay</u> = false

Allow stay.

• bool incremental evaluation = false

Incremental evaluation.

# **Additional Inherited Members**

## 5.63.1 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.

## 5.63.2 Constructor & Destructor Documentation

#### 5.63.2.1 OnePlusOneEa()

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

#### Constructor.

5.64 Pbil Class Reference 169

#### **Parameters**

```
n Size of bit vectors
```

\_mutation\_probability is initialized to 1 / n.

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

#### 5.63.3 Member Function Documentation

#### 5.63.3.1 set\_allow\_stay()

```
void set_allow_stay ( bool \ x \ ) \quad [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.

## 5.63.3.2 set\_num\_iterations()

Set the number of iterations.

#### **Parameters**

```
x Number of iterations
```

 $x \le 0$  means indefinite

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

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

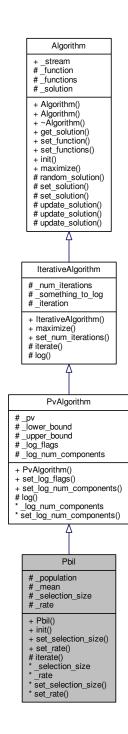
• lib/hnco/algorithms/ea/one-plus-one-ea.hh

## 5.64 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 ()

5.64 Pbil Class Reference 171

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**

## 5.64.1 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:

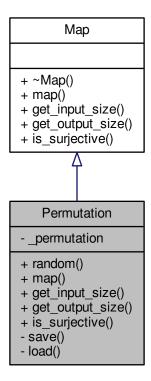
- · lib/hnco/algorithms/pv/pbil.hh
- lib/hnco/algorithms/pv/pbil.cc

# 5.65 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)

Мар.

• 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)
```

## **Private Attributes**

## **Friends**

class boost::serialization::access

## 5.65.1 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.

## 5.65.2 Member Function Documentation

```
5.65.2.1 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:

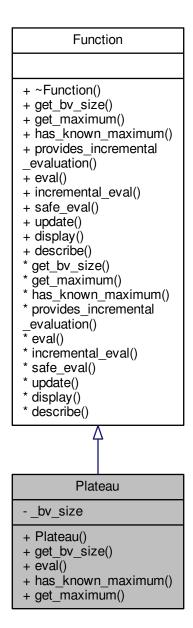
- · lib/hnco/map.hh
- lib/hnco/map.cc

## 5.66 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.

## 5.66.1 Detailed Description

Plateau.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 239 of file theory.hh.

## 5.66.2 Member Function Documentation

```
5.66.2.1 get_maximum()
```

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

Get the global maximum.

Returns

Reimplemented from Function.

Definition at line 263 of file theory.hh.

#### 5.66.2.2 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:

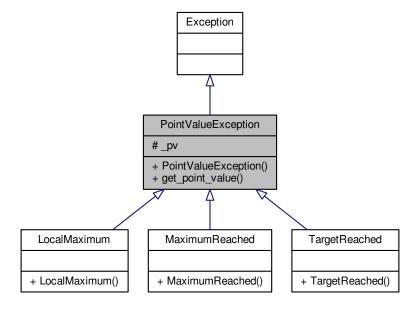
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

# 5.67 PointValueException Class Reference

Point-value exception.

#include <hnco/exception.hh>

Inheritance diagram for PointValueException:

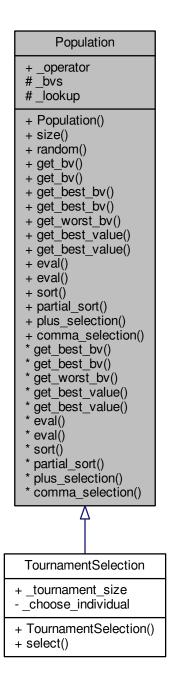


# **Public Member Functions**

PointValueException (const point_value_t &pv)
Constructor.  • const point_value_t & get_point_value () const
Get point-value.
Protected Attributes
Fiotected Attributes
<ul><li>point_value_t _pv</li></ul>
Point-value.
5.67.1 Detailed Description
5.67.1 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
5.68 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.

void partial\_sort (int selection\_size)

Partially sort the lookup table.

#### Selection

void plus\_selection (const Population &offsprings)

Plus selection.

• void comma\_selection (const Population &offsprings)

Comma selection.

#### **Public Attributes**

std::function< bool(const index\_value\_t &, const index\_value\_t &)> \_operator
 Binary operator for comparing index-value pairs.

## **Protected Attributes**

```
    std::vector< bit_vector_t > _bvs
        Bit vectors.
    std::vector< index_value_t > _lookup
        Lookup table.
```

## 5.68.1 Detailed Description

## Population.

Definition at line 35 of file population.hh.

## 5.68.2 Member Function Documentation

```
5.68.2.1 comma_selection()
```

Comma selection.

## Precondition

Offspring population must be sorted.

# Warning

The function does not break ties randomly as it should.

Definition at line 108 of file population.cc.

```
5.68.2.2 get_best_bv() [1/2] const bit_vector_t get_best_bv ( int i ) const [inline]
```

Get best bit vector.

#### **Parameters**

*i* Index in the sorted population

## Precondition

The population must be sorted.

Definition at line 89 of file population.hh.

```
5.68.2.3 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 95 of file population.hh.

```
5.68.2.4 get_best_value() [1/2] double get_best_value( int i ) const [inline]
```

Get best value.

## **Parameters**

```
i Index in the sorted population
```

## Precondition

The population must be sorted.

Definition at line 118 of file population.hh.

```
5.68.2.5 get_best_value() [2/2]
double get_best_value ( ) const [inline]
```

# Get best value.

Precondition

The population must be sorted.

Definition at line 124 of file population.hh.

#### 5.68.2.6 get\_worst\_bv()

Get worst bit vector.

#### **Parameters**

```
i Index in the sorted population
```

#### Precondition

The population must be sorted.

Definition at line 103 of file population.hh.

## 5.68.2.7 plus\_selection()

Plus selection.

## Precondition

Both populations must be sorted.

## Warning

The function does not break ties randomly as it should.

Definition at line 89 of file population.cc.

# 5.68.3 Member Data Documentation

```
5.68.3.1 _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 57 of file population.hh.

5.68.3.2 \_operator

```
std::function<bool(const index_value_t&, const index_value_t&)> _operator
```

## 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 43 of file population.hh.

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

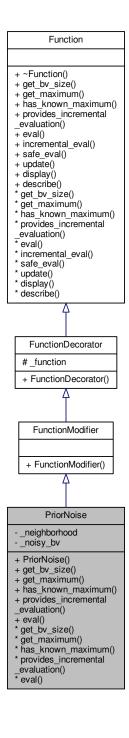
- lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

# 5.69 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**

 $\bullet \quad neighborhood {::} Neighborhood * \_neighborhood \\$ 

Neighborhood.

bit\_vector\_t \_noisy\_bv

Noisy bit vector.

## **Additional Inherited Members**

## 5.69.1 Detailed Description

Prior noise.

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

## 5.69.2 Member Function Documentation

```
5.69.2.1 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.

#### 5.69.2.2 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.

## 5.69.2.3 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

# 5.70 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.

#### **Public Attributes**

#### **Parameters**

• bool <u>log\_improvement</u> = false

Log improvement.

• std::ostream & \_stream = std::cout Output stream.

## **Protected Member Functions**

void update\_last\_improvement (double value)

Update last improvement.

## **Protected Attributes**

• Event \_last\_improvement

Last improvement.

## 5.70.1 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.

## 5.70.2 Member Function Documentation

Evaluate a bit vector.

## **Exceptions**

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

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

## 5.70.2.2 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 276 of file function-controller.hh.

#### 5.70.2.3 incremental\_eval()

Incremental evaluation.

## **Exceptions**

```
MaximumReached
TargetReached
```

Reimplemented from CallCounter.

Definition at line 172 of file function-controller.cc.

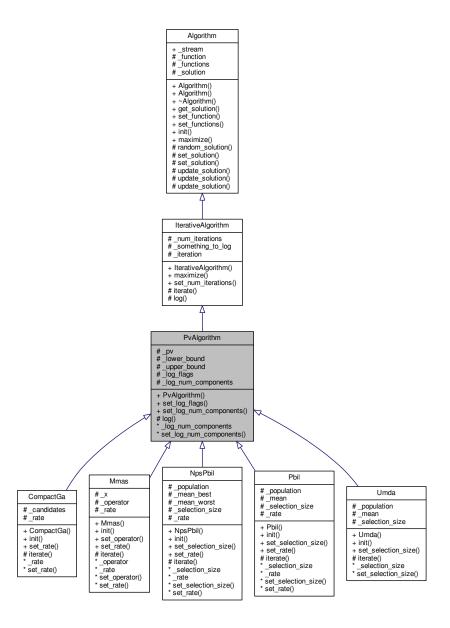
## 5.70.2.4 update()

Update after a safe evaluation.
Exceptions  MaximumReached
TargetReached
Reimplemented from CallCounter.
Definition at line 191 of file function-controller.cc.
The documentation for this class was generated from the following files:
<ul> <li>lib/hnco/functions/decorators/function-controller.hh</li> </ul>
lib/hnco/functions/decorators/function-controller.cc
5.71 PvAlgorithm Class Reference
Probability vector algorithm.
<pre>#include <hnco algorithms="" pv="" pv-algorithm.hh=""></hnco></pre>

190

**Class Documentation** 

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 <u>upper\_bound</u>

Upper bound of probability.

## **Parameters**

• int\_log\_num\_components = 5

Number of probability vector components to log.

## **Additional Inherited Members**

# 5.71.1 Detailed Description

Probability vector algorithm.

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

#### 5.71.2 Member Enumeration Documentation

## 5.71.2.1 anonymous enum

anonymous enum

## Enumerator

LOG_PV	Log probability vector.
LOG_ENTROPY	Log entropy.

5.72 Qubo Class Reference 193

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

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

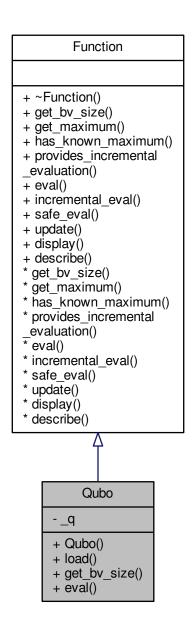
- · lib/hnco/algorithms/pv/pv-algorithm.hh
- · lib/hnco/algorithms/pv/pv-algorithm.cc

## 5.72 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.
```

#### 5.72.1 Detailed Description

Quadratic unconstrained binary optimization.

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

## 5.72.2 Member Function Documentation

Load an instance.

## **Exceptions**

Error

Definition at line 36 of file qubo.cc.

#### 5.72.3 Member Data Documentation

```
5.72.3.1 _q
```

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

Matrix.

n x n upper triangular matrix.

Definition at line 83 of file qubo.hh.

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

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

## 5.73 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

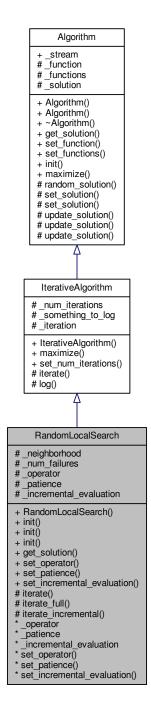
Engine.

umentation
I

5.73.1	Detailed Description
Random	numbers.
Definition	n at line 33 of file random.hh.
The doc	umentation for this struct was generated from the following files:
	/hnco/random.hh /hnco/random.cc
5.74	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\_operator (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)> \_operator = std::greater\_equal<double>()

  Binary operator for comparing evaluations.
- int \_patience = 50

Patience.

• bool \_incremental\_evaluation = false

Incremental evaluation.

## **Additional Inherited Members**

# 5.74.1 Detailed Description

Random local search.

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

### 5.74.2 Member Function Documentation

## 5.74.2.1 set\_patience()

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

Set patience.

Number of consecutive rejected moves before throwing a LocalMaximum exception

#### **Parameters**

If  $x \le 0$  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:

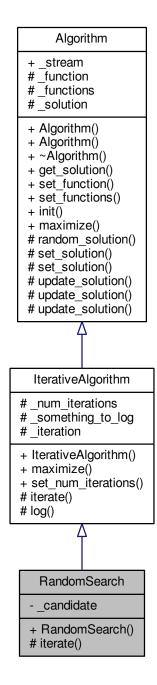
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

# 5.75 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**

## 5.75.1 Detailed Description

Random search.

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

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

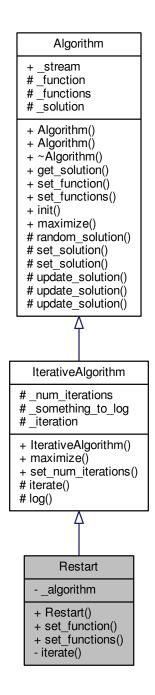
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

# 5.76 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**

## 5.76.1 Detailed Description

#### Restart.

Restart an Algorithm an indefinite number of times. Should be used in conjonction with OnBudgetFunction or StopOnMaximum.

Definition at line 38 of file restart.hh.

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

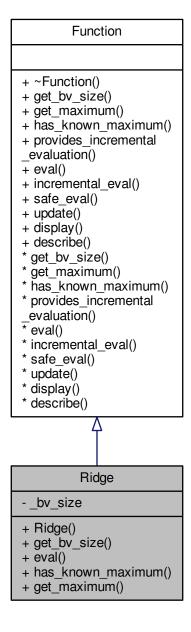
- · lib/hnco/algorithms/restart.hh
- lib/hnco/algorithms/restart.cc

# 5.77 Ridge Class Reference

## Ridge.

#include <hnco/functions/theory.hh>

Inheritance diagram for Ridge:



## **Public Member Functions**

• Ridge (int by 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.

## 5.77.1 Detailed Description

### Ridge.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 203 of file theory.hh.

#### 5.77.2 Member Function Documentation

```
5.77.2.1 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.

```
5.77.2.2 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:

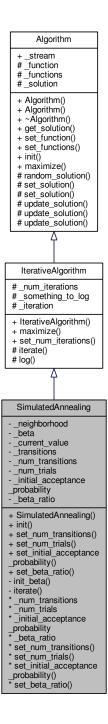
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

# 5.78 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**

# 5.78.1 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.

### 5.78.2 Member Function Documentation

## 5.78.2.1 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:

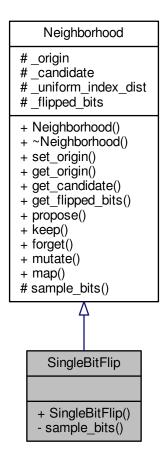
- lib/hnco/algorithms/ls/simulated-annealing.hh
- lib/hnco/algorithms/ls/simulated-annealing.cc

# 5.79 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**

## 5.79.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

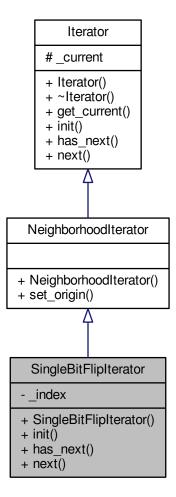
• lib/hnco/neighborhoods/neighborhood.hh

# 5.80 SingleBitFlipIterator Class Reference

Single bit flip neighborhood iterator.

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

Inheritance diagram for SingleBitFlipIterator:



# **Public Member Functions**

• SingleBitFlipIterator (int n)

Constructor.

• void init ()

Initialization.

bool has\_next ()

Has next bit vector.

void next ()

Next bit vector.

### **Private Attributes**

size\_t \_index
 Index of the last flipped bit.

## **Additional Inherited Members**

## 5.80.1 Detailed Description

Single bit flip neighborhood iterator.

Definition at line 58 of file neighborhood-iterator.hh.

#### 5.80.2 Constructor & Destructor Documentation

### 5.80.2.1 SingleBitFlipIterator()

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

Constructor.

**Parameters** 

```
n Size of bit vectors
```

Definition at line 70 of file neighborhood-iterator.hh.

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

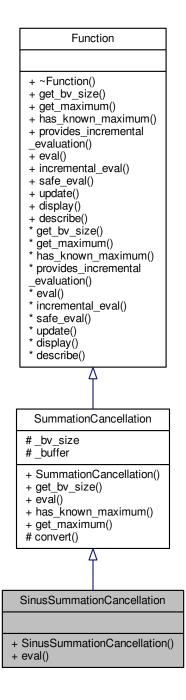
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

# 5.81 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)

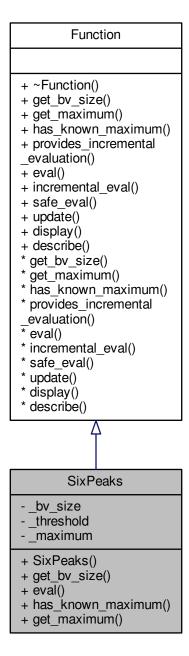
Evaluate a bit vector.

DIOL DIA GUAD TUDO TUDO TUDO
Additional Inherited Members
5.81.1 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 Comput tion. Indianapolis, 319–324.
Definition at line 103 of file cancellation.hh.
The documentation for this class was generated from the following files:
<ul> <li>lib/hnco/functions/cancellation.hh</li> <li>lib/hnco/functions/cancellation.cc</li> </ul>
5.82 SixPeaks Class Reference

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

Six Peaks.

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.

### 5.82.1 Detailed Description

Six Peaks.

It is defined by

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

#### where:

- head(x, 0) is the length of the longest prefix of x made of zeros;
- head(x, 1) is the length of the longest prefix of x made of ones;
- tail(x, 0) is the length of the longest suffix of x made of zeros;
- tail(x, 1) is the length of the longest suffix of x made of ones;
- R(x) is the reward;
- R(x) = n if (head(x, 0) > t and tail(x, 1) > t) or (head(x, 1) > t) and 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.

## 5.82.2 Member Function Documentation

```
5.82.2.1 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.
```

#### 5.82.2.2 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

# 5.83 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.

double error (const SpinMoment &target)

Compute the error.

double delta (const SpinMoment &target)

Compute the norm of the moment increment.

void sample (const SpinMoment &target, bit\_vector\_t &x)

Sample a bit vector.

#### **Public Attributes**

#### **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.

#### **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.

## 5.83.1 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.

### 5.83.2 Member Enumeration Documentation

#### 5.83.2.1 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 90 of file herding-spin.hh.

## 5.83.3 Member Function Documentation

```
5.83.3.1 q_variation()
```

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

# 5.84 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**

 $\bullet \quad \mathsf{std} :: \mathsf{vector} {<} \ \mathsf{double} > \underline{\quad} \mathsf{first}$ 

First moment.

 $\bullet \quad \mathsf{std} :: \mathsf{vector} < \mathsf{std} :: \mathsf{vector} < \mathsf{double} >> \_ \underline{\mathsf{second}}$ 

Second moment.

• double \_weight = 1

Weight of second order moments.

# 5.84.1 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:

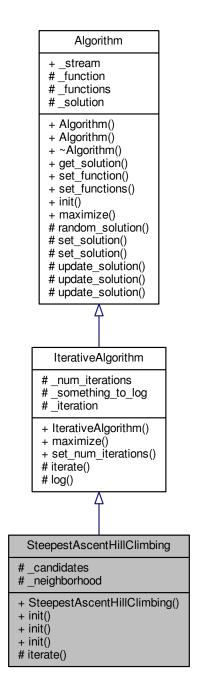
- lib/hnco/algorithms/hea/moment-spin.hh
- lib/hnco/algorithms/hea/moment-spin.cc

# 5.85 SteepestAscentHillClimbing Class Reference

Steepest ascent hill climbing.

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

Inheritance diagram for SteepestAscentHillClimbing:



## **Public Member Functions**

- $\bullet \quad Steepest Ascent Hill Climbing \ (int \ n, \ neighborhood:: Neighborhood) terator \ *neighborhood)$
- 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.

#### **Additional Inherited Members**

## 5.85.1 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:

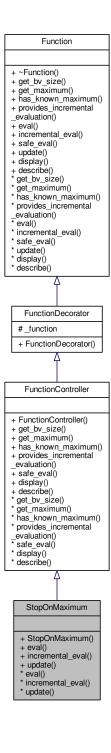
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc

# 5.86 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**

### 5.86.1 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.

#### 5.86.2 Constructor & Destructor Documentation

### 5.86.2.1 StopOnMaximum()

Constructor.

## **Parameters**

function	Decorated function
----------	--------------------

### Precondition

```
function->has_known_maximum()
```

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

## 5.86.3 Member Function Documentation

Evaluate a bit vector.

## **Exceptions**

MaximumReached

Implements Function.

Definition at line 31 of file function-controller.cc.

#### 5.86.3.2 incremental\_eval()

Incremental evaluation.

### **Exceptions**

MaximumReached

Reimplemented from Function.

Definition at line 43 of file function-controller.cc.

### 5.86.3.3 update()

Update after a safe evaluation.

#### **Exceptions**

MaximumReached

Reimplemented from Function.

Definition at line 55 of file function-controller.cc.

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

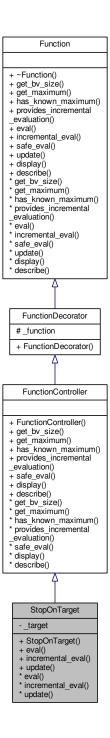
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 5.87 StopOnTarget Class Reference

Stop on target.

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

Inheritance diagram for StopOnTarget:



# **Public Member Functions**

• StopOnTarget (Function \*function, double target)

Constructor.

#### **Evaluation**

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

double incremental\_eval (const bit\_vector\_t &x, double value, const hnco::sparse\_bit\_vector\_t &flipped
bits)

Incremental evaluation.

void update (const bit\_vector\_t &x, double value)

Update after a safe evaluation.

### **Private Attributes**

• double \_target

Target.

### **Additional Inherited Members**

# 5.87.1 Detailed Description

Stop on target.

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

### 5.87.2 Constructor & Destructor Documentation

#### 5.87.2.1 StopOnTarget()

Constructor.

#### **Parameters**

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

## 5.87.3 Member Function Documentation

Evaluate a bit vector.

**Exceptions** 

TargetReached

Implements Function.

Definition at line 66 of file function-controller.cc.

## 5.87.3.2 incremental\_eval()

Incremental evaluation.

**Exceptions** 

TargetReached

Reimplemented from Function.

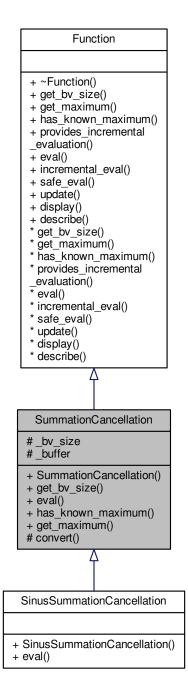
Definition at line 76 of file function-controller.cc.

## 5.87.3.3 update()

Update after a safe evaluation.

TargetReached
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
5.88 SummationCancellation Class Reference
Summation cancellation.
<pre>#include <hnco cancellation.hh="" functions=""></hnco></pre>

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.
```

### 5.88.1 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.

### 5.88.2 Constructor & Destructor Documentation

#### 5.88.2.1 SummationCancellation()

#### Constructor.

The bit vector size n must be a multiple of 9. The size of \_buffer is then n / 9.

-					
Pa	ra	m	eı	re.	rs

n Size of the bit vector

Definition at line 70 of file cancellation.hh.

## 5.88.3 Member Function Documentation

## 5.88.3.1 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:

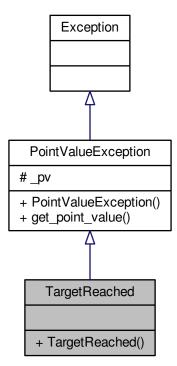
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

# 5.89 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** 

## 5.89.1 Detailed Description

target reached

Definition at line 61 of file exception.hh.

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

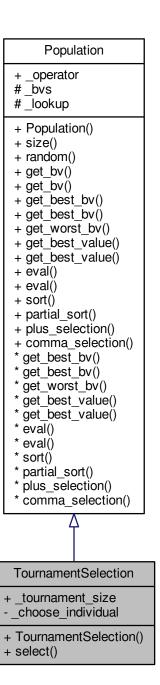
lib/hnco/exception.hh

# 5.90 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.

#### **Public Attributes**

int \_tournament\_size = 10
 Tournament size.

#### **Private Attributes**

std::uniform\_int\_distribution < int > \_choose\_individual
 Random index.

#### **Additional Inherited Members**

#### 5.90.1 Detailed Description

Population with tournament selection.

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

#### 5.90.2 Member Function Documentation

```
5.90.2.1 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.

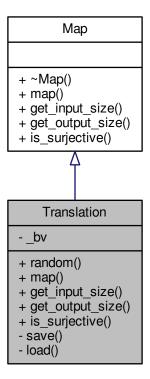
- lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

# 5.91 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)

Мар.

• 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)
```

#### **Private Attributes**

```
    bit_vector_t _bv
    Translation vector.
```

#### **Friends**

· class boost::serialization::access

#### 5.91.1 Detailed Description

#### Translation.

A translation is an affine map f from  $\mathbb{Z}_2^n$  to itself defined by f(x)=x+b, where b is an n-dimensional bit vector.

Definition at line 70 of file map.hh.

#### 5.91.2 Member Function Documentation

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

Returns

true

Reimplemented from Map.

Check for surjective map.

Definition at line 121 of file map.hh.

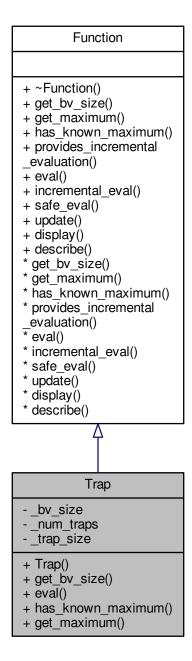
- · lib/hnco/map.hh
- lib/hnco/map.cc

# 5.92 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.

### 5.92.1 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.

#### 5.92.2 Constructor & Destructor Documentation

```
5.92.2.1 Trap()
```

#### Constructor.

#### **Parameters**

bv_size	Bit vector size
num_traps	Number of traps

Warning

bv\_size must be a multiple of num\_traps

Definition at line 64 of file trap.hh.

#### 5.92.3 Member Function Documentation

```
5.92.3.1 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.

5.92.3.2 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.

- · lib/hnco/functions/trap.hh
- lib/hnco/functions/trap.cc

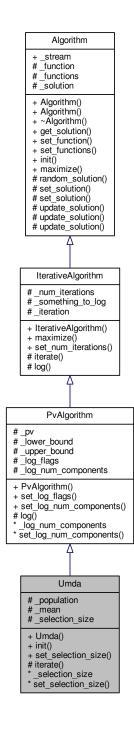
5.93 Umda Class Reference 241

### 5.93 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**

#### 5.93.1 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.

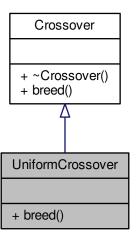
- · lib/hnco/algorithms/pv/umda.hh
- lib/hnco/algorithms/pv/umda.cc

#### 5.94 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.

#### 5.94.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

#### 5.94.2 Member Function Documentation

#### 5.94.2.1 breed()

Breed.

The offspring is the uniform crossover of two parents.

#### **Parameters**

parent1	First parent
parent2	Second parent
offspring	Offspring

h	mn	ements	Crossover.
	III		OI USSUVEI.

Definition at line 30 of file crossover.cc.

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

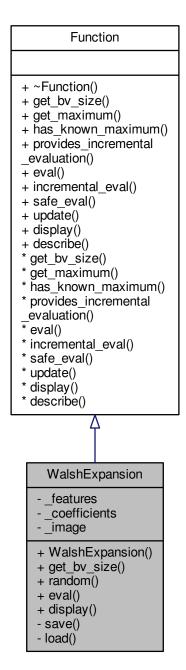
- lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

# 5.95 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.

#### **Private Member Functions**

```
    template < class Archive > void save (Archive & ar, const unsigned int version) const
```

```
    template < class Archive > void load (Archive & ar, const unsigned int version)
    Load.
```

#### **Private Attributes**

· hnco::bit\_matrix\_t \_features

Features.

• std::vector< double > \_coefficients

Coefficients.

bit\_vector\_t \_image

Image of bit vectors under the feature matrix.

#### **Friends**

· class boost::serialization::access

### 5.95.1 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.

#### 5.95.2 Member Function Documentation

#### 5.95.2.1 random()

```
void random (
          int n,
          int num_features,
          double stddev )
```

Random instance.

#### **Parameters**

n	Size of bit vector
num_features	Number of feature vectors
stddev	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

# 5.96 WalshExpansion1 Class Reference

Walsh expansion of degree 1.

#include <hnco/functions/walsh/walsh-expansion-1.hh>

Inheritance diagram for WalshExpansion1:

# Function + ~Function() + get\_bv\_size() + get\_maximum() + has\_known\_maximum() + provides\_incremental evaluation() + eval() + incremental eval() + safe eval() + update() + display() + describe() \* get\_bv\_size() \* get\_maximum() \* has\_known\_maximum() \* provides\_incremental evaluation() \* eval() \* incremental\_eval() \* safe\_eval() \* update() \* display() \* describe() WalshExpansion1 - \_linear + WalshExpansion1() + get\_bv\_size() + random() + eval() - serialize()

#### **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

#### 5.96.1 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.

#### 5.96.2 Member Function Documentation

#### 5.96.2.1 random()

Random instance.

#### **Parameters**

n	Size of bit vector
stddev	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

# 5.97 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

#include <hnco/functions/walsh/walsh-expansion-2.hh>

Inheritance diagram for WalshExpansion2:

# **Function** + ~Function() + get\_bv\_size() + get\_maximum() + has\_known\_maximum() + provides\_incremental \_evaluation() + eval() + incremental\_eval() + safe\_eval() + update() + display() + describe() \* get\_bv\_size() \* get\_maximum() \* has\_known\_maximum() \* provides\_incremental \_evaluation() \* eval() \* incremental\_eval() \* safe\_eval() \* update() \* display() describe() WalshExpansion2 - linear - \_quadratic + WalshExpansion2() + get\_bv\_size() + random() + eval() serialize()

#### **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
- std::vector< std::vector< double >> \_quadratic
   Quadratic part.

#### **Friends**

· class boost::serialization::access

#### 5.97.1 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.

#### 5.97.2 Member Function Documentation

#### 5.97.2.1 random()

```
void random (
          int n,
          double stddev_lin,
          double stddev_quad )
```

Random instance.

#### **Parameters**

n	Size of bit vector	
stddev_lin	Standard deviation of the coefficients of the linear part	
stddev_quad	Standard deviation of the coefficients of the quadratic part	

Definition at line 33 of file walsh-expansion-2.cc.

### 5.97.3 Member Data Documentation

#### 5.97.3.1 \_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.

- lib/hnco/functions/walsh/walsh-expansion-2.hh
- lib/hnco/functions/walsh/walsh-expansion-2.cc

# Index

_allow_stay	hnco::algorithm::Population, 180
hnco::neighborhood::BernoulliProcess, 34	CompactGa, 48
_expression	CompleteSearch, 51
hnco::function::MaxSat, 129	Crossover, 52
_functions	
hnco::algorithm::Algorithm, 31	DeceptiveJump, 53
_lookup	
hnco::algorithm::Population, 182	EqualProducts, 56
operator	Error, 59
hnco::algorithm::Population, 182	eval
_q	hnco::function::OnBudgetFunction, 159
hnco::function::Qubo, 195	hnco::function::ProgressTracker, 188
_quadratic	hnco::function::StopOnMaximum, 224
hnco::function::WalshExpansion2, 252	hnco::function::StopOnTarget, 228
1	Exception, 61
AdditiveGaussianNoise, 25	
AffineMap, 28	Factorization, 61
Algorithm, 30	hnco::function::Factorization, 64
	FourPeaks, 64
bernoulli_trials	Function, 67
hnco::neighborhood::MultiBitFlip, 138	FunctionController, 70
BernoulliProcess, 32	FunctionDecorator, 73
hnco::neighborhood::BernoulliProcess, 33, 34	FunctionMapComposition, 74
BiasedCrossover, 35	hnco::function::FunctionMapComposition, 76
BinaryHerding, 36	FunctionModifier, 78
BinaryMoment, 38	FunctionPlugin, 79
bit_t	hnco::function::FunctionPlugin, 81
hnco, 14	
bm_add_rows	GeneticAlgorithm, 82
hnco, 15	hnco::algorithm::GeneticAlgorithm, 84
bm_identity	get_best_bv
hnco, 15	hnco::algorithm::Population, 180, 181
bm invert	get_best_value
hnco, 15	hnco::algorithm::Population, 181
bm_multiply	get_last_improvement
hnco, 16	hnco::function::ProgressTracker, 189
bm_solve	get_maximum
hnco, 16	hnco::function::AdditiveGaussianNoise, 27
bm_solve_upper_triangular	hnco::function::DeceptiveJump, 55
hnco, 17	hnco::function::FourPeaks, 67
BmPbil, 39	hnco::function::Function, 68
breed	hnco::function::FunctionMapComposition, 77
hnco::algorithm::BiasedCrossover, 36	hnco::function::Hiff, 97
hnco::algorithm::Crossover, 53	hnco::function::Jump, 106
hnco::algorithm::UniformCrossover, 243	hnco::function::LeadingOnes, 112
,	hnco::function::Needle, 144
Cache, 43	hnco::function::Negation, 146
hnco::function::Cache, 45	hnco::function::OneMax, 162
CallCounter, 46	hnco::function::Plateau, 175
comma selection	hnco-function-PriorNoise 185

hnco::function::Ridge, 205	hnco::algorithm::Crossover
hnco::function::SixPeaks, 216	breed, 53
hnco::function::Trap, 240	hnco::algorithm::GeneticAlgorithm
get_worst_bv	GeneticAlgorithm, 84
hnco::algorithm::Population, 181	set_allow_stay, 84
Hammin a Dall OF	hnco::algorithm::IterativeAlgorithm
HammingBall, 85	IterativeAlgorithm, 101
hnco::neighborhood::HammingBall, 87	maximize, 101
HammingBallIterator, 87	set_num_iterations, 101
hnco::neighborhood::HammingBallIterator, 89	hnco::algorithm::MuCommaLambdaEa
HammingSphere, 89	MuCommaLambdaEa, 136
hnco::neighborhood::HammingSphere, 91	set_allow_stay, 136
has_known_maximum	hnco::algorithm::MuPlusLambdaEa
hnco::function::AdditiveGaussianNoise, 27	MuPlusLambdaEa, 141
hnco::function::DeceptiveJump, 55	set_allow_stay, 142
hnco::function::FourPeaks, 67 hnco::function::FunctionMapComposition, 77	hnco::algorithm::OnePlusLambdaCommaLambdaGa
hnco::function::Hiff, 97	OnePlusLambdaCommaLambdaGa, 165
hnco::function::Jump, 106	hnco::algorithm::OnePlusOneEa
hnco::function::LeadingOnes, 112	OnePlusOneEa, 168
hnco::function::LinearFunction, 114	set_allow_stay, 169
hnco::function::Needle, 144	set_num_iterations, 169
hnco::function::Negation, 147	hnco::algorithm::Population
hnco::function::OneMax, 162	_lookup, 182
hnco::function::Plateau, 175	_operator, 182
hnco::function::PriorNoise, 185	comma_selection, 180
hnco::function::Ridge, 205	get_best_bv, 180, 181
hnco::function::SixPeaks, 216	get_best_value, 181
hnco::function::SummationCancellation, 232	get_worst_bv, 181
hnco::function::Trap, 240	plus_selection, 182
•	hnco::algorithm::RandomLocalSearch
Hea< Moment, Herding >, 92	hnco::algorithm::RandomLocalSearch set_patience, 199
Hea< Moment, Herding >, 92 Hiff, 95	
Hea< Moment, Herding >, 92 Hiff, 95 hnco, 11	set_patience, 199
Hea< Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14	set_patience, 199 hnco::algorithm::SimulatedAnnealing
Hea< Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15	set_patience, 199 hnco::algorithm::SimulatedAnnealing init_beta, 208
Hea< Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15	set_patience, 199 hnco::algorithm::SimulatedAnnealing init_beta, 208 hnco::algorithm::TournamentSelection
Hea< Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20
Hea< Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16	set_patience, 199 hnco::algorithm::SimulatedAnnealing init_beta, 208 hnco::algorithm::TournamentSelection select, 235 hnco::algorithm::UniformCrossover breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18 sparse_bit_matrix_t, 14	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function, 22
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function, 22 hnco::function::AdditiveGaussianNoise
Hea< Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14  hnco::Hypercubelterator	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function, 22 hnco::function::AdditiveGaussianNoise     get_maximum, 27
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14  hnco::Hypercubelterator  next, 99	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function, 22 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14  hnco::Hypercubelterator  next, 99  hnco::Map	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function, 22 hnco::function::AdditiveGaussianNoise     get_maximum, 27
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18 sparse_bit_matrix_t, 14 sparse_bit_vector_t, 14 hnco::Hypercubelterator next, 99 hnco::Map is_surjective, 121	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18 sparse_bit_matrix_t, 14 sparse_bit_vector_t, 14 hnco::Hypercubelterator next, 99 hnco::Map is_surjective, 121 hnco::MapComposition	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function; 22 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18 sparse_bit_matrix_t, 14 sparse_bit_vector_t, 14 hnco::Hypercubelterator next, 99 hnco::Map is_surjective, 121 hnco::MapComposition is_surjective, 124	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function; 22 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14  hnco::Hypercubelterator  next, 99  hnco::Map  is_surjective, 121  hnco::MapComposition  is_surjective, 124  MapComposition, 123  hnco::Permutation  is_surjective, 173	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55
Hea < Moment, Herding >, 92  Hiff, 95  hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14  hnco::Hypercubelterator  next, 99  hnco::Map  is_surjective, 121  hnco::MapComposition  is_surjective, 124  MapComposition, 123  hnco::Permutation	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function, 22 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55     has_known_maximum, 55 has_known_maximum, 55
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18 sparse_bit_matrix_t, 14 sparse_bit_vector_t, 14 hnco::Hypercubelterator next, 99 hnco::Map is_surjective, 121 hnco::MapComposition is_surjective, 124 MapComposition, 123 hnco::Permutation is_surjective, 173 hnco::Translation is_surjective, 237	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55     has_known_maximum, 55 hnco::function::EqualProducts
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14 hnco::Hypercubelterator  next, 99 hnco::Map  is_surjective, 121 hnco::MapComposition  is_surjective, 124  MapComposition, 123 hnco::Permutation  is_surjective, 173 hnco::Translation  is_surjective, 237 hnco::algorithm, 18	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55     has_known_maximum, 55 hnco::function::EqualProducts     random, 58
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14 hnco::Hypercubelterator  next, 99 hnco::Map  is_surjective, 121 hnco::MapComposition  is_surjective, 124  MapComposition, 123 hnco::Permutation  is_surjective, 173 hnco::Translation  is_surjective, 237 hnco::algorithm, 18 hnco::algorithm::Algorithm	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55     has_known_maximum, 55 hnco::function::EqualProducts     random, 58 hnco::function::Factorization
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11 bit_t, 14 bm_add_rows, 15 bm_identity, 15 bm_invert, 15 bm_multiply, 16 bm_solve, 16 bm_solve_upper_triangular, 17 sbm_multiply, 18 sparse_bit_matrix_t, 14 sparse_bit_vector_t, 14 hnco::Hypercubelterator next, 99 hnco::Map is_surjective, 121 hnco::MapComposition is_surjective, 124 MapComposition, 123 hnco::Permutation is_surjective, 173 hnco::Translation is_surjective, 237 hnco::algorithm, 18 hnco::algorithm::Algorithm _functions, 31	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55     has_known_maximum, 55 hnco::function::EqualProducts     random, 58 hnco::function::Factorization     Factorization, 64
Hea < Moment, Herding >, 92 Hiff, 95 hnco, 11  bit_t, 14  bm_add_rows, 15  bm_identity, 15  bm_invert, 15  bm_multiply, 16  bm_solve, 16  bm_solve_upper_triangular, 17  sbm_multiply, 18  sparse_bit_matrix_t, 14  sparse_bit_vector_t, 14 hnco::Hypercubelterator  next, 99 hnco::Map  is_surjective, 121 hnco::MapComposition  is_surjective, 124  MapComposition, 123 hnco::Permutation  is_surjective, 173 hnco::Translation  is_surjective, 237 hnco::algorithm, 18 hnco::algorithm::Algorithm	set_patience, 199 hnco::algorithm::SimulatedAnnealing     init_beta, 208 hnco::algorithm::TournamentSelection     select, 235 hnco::algorithm::UniformCrossover     breed, 243 hnco::algorithm::bm_pbil, 20 hnco::algorithm::hea, 21 hnco::algorithm::hea::SpinHerding     q_variation, 218 hnco::exception, 21 hnco::function::AdditiveGaussianNoise     get_maximum, 27     has_known_maximum, 27 hnco::function::Cache     Cache, 45     provides_incremental_evaluation, 46 hnco::function::DeceptiveJump     get_maximum, 55     has_known_maximum, 55 hnco::function::EqualProducts     random, 58 hnco::function::Factorization

has been seed to the control of the	undate 100
has_known_maximum, 67 hnco::function::Function	update, 189 hnco::function::Qubo
get_maximum, 68	_q, 195
incremental_eval, 69	load, 194
provides_incremental_evaluation, 69	hnco::function::Ridge
safe_eval, 69	get_maximum, 205
hnco::function::FunctionController	has_known_maximum, 205
provides_incremental_evaluation, 72	hnco::function::SixPeaks
hnco::function::FunctionMapComposition	get_maximum, 216
FunctionMapComposition, 76	has_known_maximum, 216
get_maximum, 77	hnco::function::StopOnMaximum
has_known_maximum, 77	eval, 224
hnco::function::FunctionPlugin	incremental_eval, 225
FunctionPlugin, 81	StopOnMaximum, 224
hnco::function::Hiff	update, 225
get_maximum, 97	hnco::function::StopOnTarget
has_known_maximum, 97	eval, 228
hnco::function::Jump	incremental_eval, 228
get_maximum, 106	StopOnTarget, 227
has_known_maximum, 106	update, 228
hnco::function::LeadingOnes	hnco::function::SummationCancellation
get_maximum, 112	has_known_maximum, 232
has_known_maximum, 112	SummationCancellation, 231
hnco::function::LinearFunction	hnco::function::Trap
has_known_maximum, 114	get_maximum, 240
random, 114	has_known_maximum, 240
hnco::function::MaxSat	Trap, 239
_expression, 129	hnco::function::WalshExpansion
load, 127	random, 246
random, 128	hnco::function::WalshExpansion1
hnco::function::Needle	random, 249
get_maximum, 144	hnco::function::WalshExpansion2
has_known_maximum, 144	_quadratic, 252
hnco::function::Negation	random, 251
get maximum, 146	hnco::neighborhood, 23
has_known_maximum, 147	hnco::neighborhood::BernoulliProcess
provides_incremental_evaluation, 147	_allow_stay, 34
hnco::function::NkLandscape	BernoulliProcess, 33, 34
random, 154	set_probability, 34
hnco::function::OnBudgetFunction	hnco::neighborhood::HammingBall
eval, 159	HammingBall, 87
incremental eval, 159	hnco::neighborhood::HammingBallIterator
update, 160	HammingBallIterator, 89
hnco::function::OneMax	hnco::neighborhood::HammingSphere
get maximum, 162	HammingSphere, 91
<del>-</del> -	<b>.</b> .
has_known_maximum, 162	hnco::neighborhood::MultiBitFlip
provides_incremental_evaluation, 163	bernoulli_trials, 138
hnco::function::Plateau	MultiBitFlip, 138
get_maximum, 175	reservoir_sampling, 139
has_known_maximum, 175	hnco::neighborhood::Neighborhood
hnco::function::PriorNoise	map, 150
get_maximum, 185	mutate, 150
has_known_maximum, 185	Neighborhood, 150
provides_incremental_evaluation, 186	hnco::neighborhood::NeighborhoodIterator
hnco::function::ProgressTracker	NeighborhoodIterator, 152
eval, 188	hnco::neighborhood::SingleBitFlipIterator
get_last_improvement, 189	SingleBitFlipIterator, 211
incremental_eval, 189	hnco::random, 24

Hypercubelterator, 98	next
incremental eval	hnco::Hypercubelterator, 99
hnco::function::Function, 69	NkLandscape, 152
hnco::function::OnBudgetFunction, 159	NpsPbil, 155
hnco::function::ProgressTracker, 189	OnBudgetFunction, 158
hnco::function::StopOnMaximum, 225	OneMax, 160
hnco::function::StopOnMaximum, 223	OnePlusLambdaCommaLambdaGa, 163
init beta	hnco::algorithm::OnePlusLambdaComma
hnco::algorithm::SimulatedAnnealing, 208	LambdaGa, 165
is surjective	OnePlusOneEa, 166
hnco::Map, 121	hnco::algorithm::OnePlusOneEa, 168
hnco::MapComposition, 124	filicoalgoritimOner lasoneLa, 100
hnco::Permutation, 173	Pbil, 169
hnco::Translation, 237	Permutation, 172
IterativeAlgorithm, 100	Plateau, 174
hnco::algorithm::IterativeAlgorithm, 101	plus_selection
Iterator, 102	hnco::algorithm::Population, 182
iterator, 102	PointValueException, 176
Jump, 104	Population, 177
oump, for	PriorNoise, 183
Labs, 107	ProgressTracker, 186
LastEvaluation, 109	ProgressTracker::Event, 60
LeadingOnes, 110	provides incremental evaluation
LinearFunction, 113	hnco::function::Cache, 46
LinearMap, 116	hnco::function::Function, 69
load	hnco::function::FunctionController, 72
hnco::function::MaxSat, 127	hnco::function::Negation, 147
hnco::function::Qubo, 194	hnco::function::OneMax, 163
LocalMaximum, 118	hnco::function::PriorNoise, 186
LongPath, 119	PvAlgorithm, 190
g,	1 Vilgoriami, 100
Map, 121	q_variation
map	hnco::algorithm::hea::SpinHerding, 218
hnco::neighborhood::Neighborhood, 150	Qubo, 193
MapComposition, 122	
hnco::MapComposition, 123	Random, 195
MaxSat, 126	random
maximize	hnco::function::EqualProducts, 58
hnco::algorithm::IterativeAlgorithm, 101	hnco::function::LinearFunction, 114
MaximumReached, 124	hnco::function::MaxSat, 128
Mmas, 129	hnco::function::NkLandscape, 154
Model, 131	hnco::function::WalshExpansion, 246
ModelParameters, 132	hnco::function::WalshExpansion1, 249
MuCommaLambdaEa, 134	hnco::function::WalshExpansion2, 251
hnco::algorithm::MuCommaLambdaEa, 136	RandomLocalSearch, 196
MuPlusLambdaEa, 139	RandomSearch, 199
hnco::algorithm::MuPlusLambdaEa, 141	reservoir_sampling
MultiBitFlip, 137	hnco::neighborhood::MultiBitFlip, 139
hnco::neighborhood::MultiBitFlip, 138	Restart, 201
mutate	Ridge, 203
hnco::neighborhood::Neighborhood, 150	
	safe_eval
Needle, 142	hnco::function::Function, 69
Negation, 145	sbm_multiply
Neighborhood, 148	hnco, 18
hnco::neighborhood::Neighborhood, 150	select
NeighborhoodIterator, 151	hnco::algorithm::TournamentSelection, 235
hnco::neighborhood::NeighborhoodIterator, 152	set_allow_stay

```
hnco::algorithm::GeneticAlgorithm, 84
     hnco::algorithm::MuCommaLambdaEa, 136
    hnco::algorithm::MuPlusLambdaEa, 142
     hnco::algorithm::OnePlusOneEa, 169
set_num_iterations
    hnco::algorithm::IterativeAlgorithm, 101
    hnco::algorithm::OnePlusOneEa, 169
set patience
     hnco::algorithm::RandomLocalSearch, 199
set probability
     hnco::neighborhood::BernoulliProcess, 34
SimulatedAnnealing, 206
SingleBitFlip, 208
SingleBitFlipIterator, 210
     hnco::neighborhood::SingleBitFlipIterator, 211
SinusSummationCancellation, 211
SixPeaks, 213
sparse_bit_matrix_t
     hnco, 14
sparse_bit_vector_t
    hnco, 14
SpinHerding, 216
SpinMoment, 219
SteepestAscentHillClimbing, 220
StopOnMaximum, 222
     hnco::function::StopOnMaximum, 224
StopOnTarget, 226
    hnco::function::StopOnTarget, 227
SummationCancellation, 229
    hnco::function::SummationCancellation, 231
TargetReached, 232
TournamentSelection, 234
Translation, 236
Trap, 238
     hnco::function::Trap, 239
Umda, 241
UniformCrossover, 243
update
     hnco::function::OnBudgetFunction, 160
     hnco::function::ProgressTracker, 189
     hnco::function::StopOnMaximum, 225
    hnco::function::StopOnTarget, 228
WalshExpansion, 244
WalshExpansion1, 247
WalshExpansion2, 250
```