HNCO

0.9

Generated by Doxygen 1.8.13

# **Contents**

1	Nam	espace	Index		1
	1.1	Names	space List		. 1
2	Hier	archical	Index		3
	2.1	Class I	Hierarchy		. 3
3	Clas	s Index			7
	3.1	Class I	_ist		. 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	. 15
			4.1.2.3	sparse_bit_vector_t	. 15
		4.1.3	Function	Documentation	. 15
			4.1.3.1	bm_add_rows()	. 15
			4.1.3.2	bm_identity()	. 16
			4.1.3.3	bm_invert()	. 16
			4.1.3.4	bm_multiply()	. 17
			4.1.3.5	bm_rank()	. 17
			4.1.3.6	bm_row_echelon_form()	. 17
			4.1.3.7	bm_solve()	. 17
			4.1.3.8	bm solve upper triangular()	. 18

ii CONTENTS

			4.1.3.9 bv_from_vector_bool()	19
			4.1.3.10 bv_to_vector_bool()	19
			4.1.3.11 sbm_multiply()	19
	4.2	hnco::a	Igorithm Namespace Reference	20
		4.2.1	Detailed Description	22
	4.3	hnco::a	lgorithm::bm_pbil Namespace Reference	22
		4.3.1	Detailed Description	22
	4.4	hnco::a	Igorithm::hea Namespace Reference	22
		4.4.1	Detailed Description	23
	4.5	hnco::e	xception Namespace Reference	23
		4.5.1	Detailed Description	23
	4.6	hnco::fu	unction Namespace Reference	23
		4.6.1	Detailed Description	25
	4.7	hnco::n	eighborhood Namespace Reference	25
		4.7.1	Detailed Description	26
	4.8	hnco::ra	andom Namespace Reference	26
		4.8.1	Detailed Description	26
5	Clas	s Docun	nentation	27
	5.1		eGaussianNoise Class Reference	
	0.1	5.1.1	Detailed Description	28
			Member Function Documentation	28
			5.1.2.1 get_maximum()	28
			5.1.2.2 has known maximum()	28
	5.2		ap Class Reference	29
	5.2	5.2.1	Detailed Description	30
			Member Function Documentation	30
			5.2.2.1 is_surjective()	30
	<b>.</b>	A1	5.2.2.2 random()	30
	5.3	_	m Class Reference	31
		5.3.1	Detailed Description	33

CONTENTS

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

iv CONTENTS

	5.9.2	Constructor & Destructor Documentation	45
		5.9.2.1 Cache()	45
	5.9.3	Member Function Documentation	45
		5.9.3.1 provides_incremental_evaluation()	46
5.10	CallCo	unter Class Reference	46
	5.10.1	Detailed Description	47
5.11	Compa	actGa Class Reference	47
	5.11.1	Detailed Description	49
5.12	Comple	eteSearch Class Reference	49
	5.12.1	Detailed Description	50
5.13	Crosso	over Class Reference	50
	5.13.1	Detailed Description	50
	5.13.2	Member Function Documentation	51
		5.13.2.1 breed()	51
5.14	Decept	tiveJump Class Reference	51
	5.14.1	Detailed Description	52
	5.14.2	Member Function Documentation	52
		5.14.2.1 get_maximum()	52
		5.14.2.2 has_known_maximum()	53
5.15	EqualP	Products Class Reference	53
	5.15.1	Detailed Description	54
	5.15.2	Member Function Documentation	54
		5.15.2.1 random()	54
5.16	Error C	Class Reference	55
	5.16.1	Detailed Description	56
5.17	Progres	ssTracker::Event Struct Reference	56
	5.17.1	Detailed Description	56
5.18	Except	tion Class Reference	56
	5.18.1	Detailed Description	57
5.19	Factoria	zation Class Reference	57

CONTENTS

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

vi

		5.25.3.2 has_known_maximum()	71
5.26	Functio	onModifier Class Reference	72
	5.26.1	Detailed Description	72
5.27	Functio	onPlugin Class Reference	73
	5.27.1	Detailed Description	74
	5.27.2	Constructor & Destructor Documentation	74
		5.27.2.1 FunctionPlugin()	74
5.28	Genetic	cAlgorithm Class Reference	74
	5.28.1	Detailed Description	76
	5.28.2	Constructor & Destructor Documentation	76
		5.28.2.1 GeneticAlgorithm()	76
	5.28.3	Member Function Documentation	77
		5.28.3.1 set_allow_stay()	77
5.29	Hammi	ngBall Class Reference	77
	5.29.1	Detailed Description	78
	5.29.2	Constructor & Destructor Documentation	78
		5.29.2.1 HammingBall()	78
5.30	Hammi	ngSphere Class Reference	79
	5.30.1	Detailed Description	80
	5.30.2	Constructor & Destructor Documentation	80
		5.30.2.1 HammingSphere()	80
5.31	Hammi	ngSphereIterator Class Reference	80
	5.31.1	Detailed Description	81
	5.31.2	Constructor & Destructor Documentation	82
		5.31.2.1 HammingSphereIterator()	82
5.32	Hboa C	Class Reference	82
	5.32.1	Detailed Description	83
5.33	Hea<	Moment, Herding > Class Template Reference	83
	5.33.1	Detailed Description	86
	5.33.2	Member Enumeration Documentation	86

CONTENTS vii

		5.33.2.1 anonymous enum	86
		5.33.2.2 anonymous enum	86
	5.33.3	Constructor & Destructor Documentation	87
		5.33.3.1 Hea()	87
	5.33.4	Member Function Documentation	87
		5.33.4.1 set_reset_period()	87
		5.33.4.2 set_selection_size()	87
5.34	Hiff Cla	ass Reference	88
	5.34.1	Detailed Description	89
	5.34.2	Member Function Documentation	89
		5.34.2.1 get_maximum()	89
		5.34.2.2 has_known_maximum()	89
5.35	HncoE	valuator Class Reference	90
	5.35.1	Detailed Description	90
5.36	Hypero	subelterator Class Reference	91
	5.36.1	Detailed Description	91
5.37	Iterative	eAlgorithm Class Reference	92
	5.37.1	Detailed Description	93
	5.37.2	Constructor & Destructor Documentation	93
		5.37.2.1 IterativeAlgorithm()	93
	5.37.3	Member Function Documentation	93
		5.37.3.1 maximize()	93
		5.37.3.2 set_num_iterations()	94
5.38	Iterator	Class Reference	94
	5.38.1	Detailed Description	95
5.39	Jump (	Class Reference	96
	5.39.1	Detailed Description	96
	5.39.2	Member Function Documentation	97
		5.39.2.1 get_maximum()	97
		5.39.2.2 has_known_maximum()	97

viii CONTENTS

5.40	Labs Cla	ass Refer	ence			 	 	 	 	 	 	97
	5.40.1	Detailed [	Description			 	 	 	 	 	 	98
5.41	LastEva	luation Cl	ass Referen	ice		 	 	 	 	 	 	98
	5.41.1	Detailed [	Description			 	 	 	 	 	 	99
5.42	Leading	Ones Cla	ss Referenc	е		 	 	 	 	 	 	99
	5.42.1	Detailed [	Description			 	 	 	 	 	 	100
	5.42.2	Member F	Function Do	cumentat	ion .	 	 	 	 	 	 	100
		5.42.2.1	get_maxim	um()		 	 	 	 	 	 	100
		5.42.2.2	has_known	ı_maximu	ım() .	 	 	 	 	 	 	100
5.43	LinearF	unction Cl	ass Referer	nce		 	 	 	 	 	 	101
	5.43.1	Detailed [	Description			 	 	 	 	 	 	102
	5.43.2	Member F	unction Do	cumentati	ion .	 	 	 	 	 	 	102
		5.43.2.1	has_knowr	ı_maximu	ım() .	 	 	 	 	 	 	102
		5.43.2.2	random() .			 	 	 	 	 	 	102
5.44	LinearM	lap Class	Reference .			 	 	 	 	 	 	103
	5.44.1	Detailed [	Description			 	 	 	 	 	 	104
	5.44.2	Member F	Function Do	cumentati	ion .	 	 	 	 	 	 	104
		5.44.2.1	is_surjectiv	re()		 	 	 	 	 	 	104
		5.44.2.2	random() .			 	 	 	 	 	 	104
5.45	LocalMa	aximum C	lass Referer	nce		 	 	 	 	 	 	105
	5.45.1	Detailed [	Description			 	 	 	 	 	 	105
5.46	LongPat	th Class F	Reference .			 	 	 	 	 	 	106
	5.46.1	Detailed [	Description			 	 	 	 	 	 	106
5.47	Ltga Cla	ass Refere	ence			 	 	 	 	 	 	107
	5.47.1	Detailed [	Description			 	 	 	 	 	 	108
5.48	Map Cla	ass Refere	ence			 	 	 	 	 	 	108
	5.48.1	Detailed [	Description			 	 	 	 	 	 	109
	5.48.2	Member F	Function Do	cumentat	ion .	 	 	 	 	 	 	109
		5.48.2.1	is_surjectiv	re()		 	 	 	 	 	 	109
5.49	MapCor	mposition	Class Refer	ence		 	 	 	 	 	 	109

CONTENTS

	5.49.1	Detailed Description	10
	5.49.2	Constructor & Destructor Documentation	10
		5.49.2.1 MapComposition()	10
	5.49.3	Member Function Documentation	11
		5.49.3.1 is_surjective()	11
5.50	Maxim	umReached Class Reference	11
	5.50.1	Detailed Description	12
5.51	MaxSa	t Class Reference	12
	5.51.1	Detailed Description	13
	5.51.2	Member Function Documentation	13
		5.51.2.1 load()	13
		5.51.2.2 random() [1/2]	13
		5.51.2.3 random() [2/2] 1	14
	5.51.3	Member Data Documentation	14
		5.51.3.1 _expression	14
5.52	Mmas	Class Reference	15
	5.52.1	Detailed Description	16
5.53	Model	Class Reference	16
	5.53.1	Detailed Description	17
5.54	ModelF	Parameters Class Reference	17
	5.54.1	Detailed Description	18
5.55	MuCor	nmaLambdaEa Class Reference	18
	5.55.1	Detailed Description	19
	5.55.2	Constructor & Destructor Documentation	19
		5.55.2.1 MuCommaLambdaEa()	19
	5.55.3	Member Function Documentation	20
		5.55.3.1 set_allow_stay()	20
5.56	MultiBi	tFlip Class Reference	20
	5.56.1	Detailed Description	21
	5.56.2	Constructor & Destructor Documentation	21

CONTENTS

		5.56.2.1 MultiBitFlip()	121
	5.56.3	Member Function Documentation	121
		5.56.3.1 bernoulli_trials()	121
		5.56.3.2 reservoir_sampling()	122
5.57	MuPlus	sLambdaEa Class Reference	122
	5.57.1	Detailed Description	123
	5.57.2	Constructor & Destructor Documentation	124
		5.57.2.1 MuPlusLambdaEa()	124
	5.57.3	Member Function Documentation	124
		5.57.3.1 set_allow_stay()	124
5.58	Needle	Class Reference	125
	5.58.1	Detailed Description	125
	5.58.2	Member Function Documentation	126
		5.58.2.1 get_maximum()	126
		5.58.2.2 has_known_maximum()	126
5.59	Negation	on Class Reference	127
	5.59.1	Detailed Description	128
	5.59.2	Member Function Documentation	128
		5.59.2.1 get_maximum()	128
		5.59.2.2 has_known_maximum()	128
		5.59.2.3 provides_incremental_evaluation()	129
5.60	Neighb	orhood Class Reference	129
	5.60.1	Detailed Description	131
	5.60.2	Constructor & Destructor Documentation	131
		5.60.2.1 Neighborhood()	131
	5.60.3	Member Function Documentation	131
		5.60.3.1 map()	131
		5.60.3.2 mutate()	132
5.61	Neighb	orhoodIterator Class Reference	132
	5.61.1	Detailed Description	133

CONTENTS xi

	5.61.2	Constructor & Destructor Documentation	33
		5.61.2.1 NeighborhoodIterator()	33
5.62	NkLand	dscape Class Reference	33
	5.62.1	Detailed Description	34
	5.62.2	Member Function Documentation	35
		5.62.2.1 random()	35
5.63	NpsPbi	il Class Reference	35
	5.63.1	Detailed Description	37
5.64	OnBud	getFunction Class Reference	37
	5.64.1	Detailed Description	39
	5.64.2	Member Function Documentation	39
		5.64.2.1 eval()	39
		5.64.2.2 incremental_eval()	39
		5.64.2.3 update()	40
5.65	OneMa	ax Class Reference	40
	5.65.1	Detailed Description	41
	5.65.2	Member Function Documentation	41
		5.65.2.1 get_maximum()	41
		5.65.2.2 has_known_maximum()	42
		5.65.2.3 provides_incremental_evaluation()	42
5.66	OnePlu	usLambdaCommaLambdaGa Class Reference	42
	5.66.1	Detailed Description	43
	5.66.2	Constructor & Destructor Documentation	44
		5.66.2.1 OnePlusLambdaCommaLambdaGa()	44
5.67	OnePlu	usOneEa Class Reference	44
	5.67.1	Detailed Description	45
	5.67.2	Constructor & Destructor Documentation	46
		5.67.2.1 OnePlusOneEa()	46
	5.67.3	Member Function Documentation	46
		5.67.3.1 set_allow_stay()	46

xii CONTENTS

		5.67.3.2	set_num_ite	rations() .			 	 	 	 	146
5.68	Parame	eterlessPop	ulationPyran	nid Class R	eference	·	 	 	 	 	147
	5.68.1	Detailed D	escription				 	 	 	 	147
5.69	Pbil Cla	ass Referer	nce				 	 	 	 	148
	5.69.1	Detailed D	escription				 	 	 	 	149
5.70	Permut	ation Class	Reference				 	 	 	 	149
	5.70.1	Detailed D	escription				 	 	 	 	150
	5.70.2	Member F	unction Docu	umentation			 	 	 	 	150
		5.70.2.1	is_surjective	()			 	 	 	 	151
5.71	Plateau	ı Class Ref	erence				 	 	 	 	151
	5.71.1	Detailed D	escription				 	 	 	 	152
	5.71.2	Member F	unction Docu	umentation			 	 	 	 	152
		5.71.2.1	get_maximu	m()			 	 	 	 	152
		5.71.2.2	has_known_	_maximum()	)		 	 	 	 	152
5.72	PointVa	alueExcepti	on Class Ref	erence .			 	 	 	 	153
	5.72.1	Detailed D	escription				 	 	 	 	153
5.73	Popula	tion Class F	Reference .				 	 	 	 	154
	5.73.1	Detailed D	escription				 	 	 	 	155
	5.73.2	Member F	unction Docu	umentation			 	 	 	 	155
		5.73.2.1	comma_sele	ection()			 	 	 	 	155
		5.73.2.2	get_best_bv	<b>()</b> [1/2] .			 	 	 	 	156
		5.73.2.3	get_best_bv	<b>()</b> [2/2] .			 	 	 	 	156
		5.73.2.4	get_best_va	lue() [1/2]			 	 	 	 	156
		5.73.2.5	get_best_va	lue() [2/2]			 	 	 	 	157
		5.73.2.6	get_worst_b	v()			 	 	 	 	157
		5.73.2.7	plus_selection	on()			 	 	 	 	157
	5.73.3	Member D	ata Docume	ntation			 	 	 	 	158
		5.73.3.1	_compare_ir	ndex_value			 	 	 	 	158
		5.73.3.2	_lookup				 	 	 	 	158
5.74	PriorNo	oise Class F	Reference .				 	 	 	 	159

CONTENTS xiii

	5.74.1	Detailed Description	30
	5.74.2	Member Function Documentation	30
		5.74.2.1 get_maximum()	30
		5.74.2.2 has_known_maximum()	30
		5.74.2.3 provides_incremental_evaluation()	31
5.75	Progres	ssTracker Class Reference	31
	5.75.1	Detailed Description	62
	5.75.2	Member Function Documentation	63
		5.75.2.1 eval()	63
		5.75.2.2 get_last_improvement()	63
		5.75.2.3 incremental_eval()	63
		5.75.2.4 update()	64
5.76	PvAlgo	rithm Class Reference	64
	5.76.1	Detailed Description	35
	5.76.2	Member Enumeration Documentation	35
		5.76.2.1 anonymous enum	35
5.77	Qubo C	Class Reference	36
	5.77.1	Detailed Description	37
	5.77.2	Member Function Documentation	37
		5.77.2.1 load()	37
	5.77.3	Member Data Documentation	38
		5.77.3.1 _q	38
5.78	Randor	m Struct Reference	38
	5.78.1	Detailed Description	38
5.79	Randor	mLocalSearch Class Reference	69
	5.79.1	Detailed Description	70
	5.79.2	Member Function Documentation	70
		5.79.2.1 set_patience()	70
5.80	Randor	mSearch Class Reference	71
	5.80.1	Detailed Description	72

xiv CONTENTS

5.81	Restart	t Class Reference	72
	5.81.1	Detailed Description	73
5.82	Ridge (	Class Reference	73
	5.82.1	Detailed Description	74
	5.82.2	Member Function Documentation	74
		5.82.2.1 get_maximum()	74
		5.82.2.2 has_known_maximum()	75
5.83	Simula	tedAnnealing Class Reference	75
	5.83.1	Detailed Description	77
	5.83.2	Member Function Documentation	77
		5.83.2.1 init_beta()	77
5.84	Single	BitFlip Class Reference	77
	5.84.1	Detailed Description	78
5.85	Single	BitFlipIterator Class Reference	78
	5.85.1	Detailed Description	79
	5.85.2	Constructor & Destructor Documentation	79
		5.85.2.1 SingleBitFlipIterator()	79
5.86	SinusS	ummationCancellation Class Reference	80
	5.86.1	Detailed Description	80
5.87	SixPea	ks Class Reference	81
	5.87.1	Detailed Description	82
	5.87.2	Member Function Documentation	82
		5.87.2.1 get_maximum()	82
		5.87.2.2 has_known_maximum()	83
5.88	SpinHe	erding Class Reference	83
	5.88.1	Detailed Description	84
	5.88.2	Member Enumeration Documentation	85
		5.88.2.1 anonymous enum	85
	5.88.3	Constructor & Destructor Documentation	85
		5.88.3.1 SpinHerding()	85

CONTENTS xv

	5.88.4	Member Function Documentation	85
		5.88.4.1 q_variation()	85
5.89	SpinMo	oment Struct Reference	86
	5.89.1	Detailed Description	87
5.90	Steepe	stAscentHillClimbing Class Reference	87
	5.90.1	Detailed Description	88
5.91	StopOr	nMaximum Class Reference	88
	5.91.1	Detailed Description	89
	5.91.2	Constructor & Destructor Documentation	89
		5.91.2.1 StopOnMaximum()	89
	5.91.3	Member Function Documentation	90
		5.91.3.1 eval()	90
		5.91.3.2 incremental_eval()	90
		5.91.3.3 update()	90
5.92	StopOr	nTarget Class Reference	91
	5.92.1	Detailed Description	92
	5.92.2	Constructor & Destructor Documentation	92
		5.92.2.1 StopOnTarget()	92
	5.92.3	Member Function Documentation	92
		5.92.3.1 eval()	92
		5.92.3.2 incremental_eval()	93
		5.92.3.3 update()	93
5.93	Summa	ationCancellation Class Reference	93
	5.93.1	Detailed Description	95
	5.93.2	Constructor & Destructor Documentation	95
		5.93.2.1 SummationCancellation()	95
	5.93.3	Member Function Documentation	95
		5.93.3.1 has_known_maximum()	95
5.94	TargetF	Reached Class Reference	96
	5.94.1	Detailed Description	96

xvi CONTENTS

Index		213
	5.103.2.1 feature	211
	5.103.2 Member Data Documentation	
	5.103.1 Detailed Description	
5.103	3Function::WalshTransformTerm Struct Reference	
	5.102.3.1 _quadratic	
	5.102.3 Member Data Documentation	
	5.102.2.1 random()	
	5.102.2 Member Function Documentation	
	5.102.1 Detailed Description	
5.102	2WalshExpansion2 Class Reference	209
	5.101.2.1 random()	208
	5.101.2 Member Function Documentation	208
	5.101.1 Detailed Description	208
5.101	1WalshExpansion1 Class Reference	207
	5.100.2.1 random()	206
	5.100.2 Member Function Documentation	206
	5.100.1 Detailed Description	206
5.100	0WalshExpansion Class Reference	205
	5.99.2.1 breed()	205
	5.99.2 Member Function Documentation	205
	5.99.1 Detailed Description	204
5.99	UniformCrossover Class Reference	204
	5.98.1 Detailed Description	204
5.98	Umda Class Reference	202
	5.97.3.2 has_known_maximum()	
	5.97.3.1 get_maximum()	
	5.97.3 Member Function Documentation	
	5.97.2.1 Trap()	
	5.97.2 Constructor & Destructor Documentation	
3.37	5.97.1 Detailed Description	
5.07	Trap Class Reference	
	5.96.2.1 is_surjective()	
	5.96.1 Detailed Description	
5.96	Translation Class Reference	
F 00	5.95.2.1 select()	
	5.95.2 Member Function Documentation	
	5.95.1 Detailed Description	
5.95		
5.95	TournamentSelection Class Reference	197

### **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	20
hnco::algorithm::bm_pbil	
Boltzmann machine PBIL	22
hnco::algorithm::hea	
Herding evolutionary algorithm	22
hnco::exception	
Exceptions	23
hnco::function	
Functions to be maximized	23
hnco::neighborhood	
Neighborhoods for local search	25
hnco::random	
Pseudo random numbers	26

2 Namespace Index

# **Chapter 2**

# **Hierarchical Index**

### 2.1 Class Hierarchy

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

Algorithm	1
CompleteSearch	
Hboa	
Ltga	
ParameterlessPopulationPyramid	
IterativeAlgorithm	
BmPbil	
FirstAscentHillClimbing	
GeneticAlgorithm	
Hea< Moment, Herding >	
MuCommaLambdaEa	
MuPlusLambdaEa	
OnePlusLambdaCommaLambdaGa14	
PvAlgorithm	
CompactGa	.7
Mmas	5
NpsPbil	5
Pbil	8
Umda	2
RandomLocalSearch	9
RandomSearch	1
Restart	2
SimulatedAnnealing	5
SteepestAscentHillClimbing	7
OnePlusOneEa	4
BinaryHerding	7
BinaryMoment	
Crossover	
BiasedCrossover	Ē
	-
UniformCrossover	4
Evaluator	
HncoEvaluator	
ProgressTracker::Event	
Exception	6

4 Hierarchical Index

Error	 	 	 	 	 55
LastEvaluation	 	 	 	 	 98
PointValueException	 	 	 	 	 153
LocalMaximum	 	 	 	 	 105
MaximumReached					
TargetReached					
Function					
DeceptiveJump					
EqualProducts					
Factorization					
FourPeaks					
FunctionDecorator					
FunctionController	 	 	 	 	 67
Cache	 	 	 	 	 44
CallCounter	 	 	 	 	 46
OnBudgetFunction	 	 	 	 	 137
ProgressTracker					
StopOnMaximum					
StopOnTarget					
FunctionModifier					
AdditiveGaussianNoise					
FunctionMapComposition .					
Negation					
PriorNoise					
FunctionPlugin					
Hiff					
Jump	 	 	 	 	 96
Labs					
LeadingOnes	 	 	 	 	 99
LinearFunction	 	 	 	 	 101
LongPath	 	 	 	 	 106
MaxSat	 	 	 	 	 112
Needle	 	 	 	 	 125
NkLandscape	 	 	 	 	 133
OneMax	 	 	 	 	 140
Plateau	 	 	 	 	 151
Qubo					
Ridge	 	 	 	 	 173
SixPeaks					
SummationCancellation					
SinusSummationCancellation .					
Trap					
WalshExpansion					
WalshExpansion1					
WalshExpansion2					
•					
Iterator					
Hypercubelterator					
NeighborhoodIterator	 	 	 	 	 132
HammingSphereIterator	 	 	 	 	 80
SingleBitFlipIterator	 	 	 	 	 178
Map	 	 	 	 	 108
AffineMap					
LinearMap					
MapComposition					
Permutation					
Translation					
Model	 	 	 	 	 116

2.1 Class Hierarchy 5

ModelParameters
Neighborhood
MultiBitFlip
BernoulliProcess
HammingBall
HammingSphere
SingleBitFlip
Population
TournamentSelection
Random
SpinHerding
SpinMoment
Function::WalshTransformTerm

6 Hierarchical Index

### **Chapter 3**

# **Class Index**

### 3.1 Class List

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

AdditiveGaussianivoise	
Additive Gaussian Noise	27
AffineMap	
Affine map	29
Algorithm	
Abstract search algorithm	31
BernoulliProcess	
Bernoulli process	33
BiasedCrossover	
Biased crossover	36
BinaryHerding	
Herding with binary variables	37
BinaryMoment	
Moment for binary variables	39
BmPbil	
Boltzmann machine PBIL	40
Cache	
Cache	44
CallCounter	
Call counter	46
CompactGa	
Compact genetic algorithm	47
CompleteSearch	40
Complete search	49
Crossover	E0
Crossover	50
Deceptive jump	51
Deceptive jump	51
Equal products	53
Error	55
Error	55
ProgressTracker::Event	55
Event	56
Exception	50
Rasic excention	56

8 Class Index

Factorization		
		57
FirstAscentHillC		
	scent hill climbing	59
FourPeaks	eaks	80
Function	edns	JU
	on	33
FunctionContro		_
Functi	on controller	67
FunctionDecora		
		8
FunctionMapCo	·	
	· ·	89
FunctionModifie		70
FunctionPlugin	on modifier	'2
	on plugin	73
GeneticAlgorith		Ü
		<b>7</b> 4
HammingBall		
Hamn	ing ball	7
HammingSpher		
		<b>7</b> 9
HammingSpher		
	ing sphere neighborhood iterator	30
Hboa	shipal Dayasian Ontimization Algorithm	20
Hea< Moment,	, ,	32
		33
Hiff	g ovoidilonary digonium	,0
Hierar	chical if and only if	88
HncoEvaluator		
Evalua	tor for HNCO functions	90
Hypercubeltera		
• •		91
IterativeAlgorith		
	e search	92
Iterator	r over bit vectors	94
Jump	Tover bit vectors	14
•		96
Labs	•	•
	utocorrelation binary sequences	97
LastEvaluation		
Last e	aluation	8
LeadingOnes		
	g ones 9	99
LinearFunction	· · · ·	
	function	)1
LinearMap	map	าว
LocalMaximum	шар	IJ
	maximum	)5
LongPath		_
	path	)6
Ltga		
	e Tree Genetic Algorithm	)7

3.1 Class List

Мар		
	The state of the s	108
MapCom		109
Maximur	mReached	103
		111
MaxSat	MAX-SAT	112
Mmas	WAA-OAI	112
	Max-min ant system	115
Model	Model of a Deltamana machine	110
ModelPa	Model of a Boltzmann machine	116
	Parameters of a Boltzmann machine	117
MuComr	maLambdaEa	
MultiBitF		118
Manubiti	•	120
MuPlusL	ambdaEa	
Noodlo	(mu+lambda) EA	122
Needle	Needle in a haystack	125
Negation	•	
	-3	127
Neighbo		129
Neighbo	rhoodIterator	120
	- 3	132
NkLands	•	133
NpsPbil	TWO failuscape	100
•	Population-based incremental learning with negative and positive selection	135
OnBudge	etFunction	107
OneMax		137
		140
OnePlus	LambdaCommaLambdaGa	
OnePlus	(1+(lambda, lambda)) genetic algorithm	142
Office las	(1+1) EA	144
Paramet	erlessPopulationPyramid	
Dhil	Parameter-less Population Pyramid	147
Pbil	Population-based incremental learning	148
Permuta	tion	
	Permutation	149
Plateau	Plateau	151
PointValu	ueException	101
	Point-value exception	153
Population		151
PriorNois	·	154
	Prior noise	159
Progress		464
PvAlgori	ProgressTracker	161
	Probability vector algorithm	164

10 Class Index

Qubo		
	Quadratic unconstrained binary optimization	166
Random		
	Random numbers	68
	ocalSearch Random local search	169
RandomS		00
F	Random search	71
Restart		. 70
Ridge	Restart	/2
	Ridge	173
Simulated		
	Simulated annealing	75
SingleBitF		. 77
SingleBitF	One bit neighborhood	11
	Single bit flip neighborhood iterator	78
	mationCancellation	_
	Summation cancellation with sinus	80
SixPeaks		
SpinHerdi	Six Peaks	81
	Herding with spin variables	183
SpinMome		-
•	Moment for spin variables	86
	scentHillClimbing	
	Steepest ascent hill climbing	87
StopOnMa	aximum Stop on maximum     .  .  .	98
StopOnTa	·	00
	Stop on target	91
Summatio	nCancellation	
	Summation cancellation	93
TargetRea		IOE
	Target reached	30
	Population with tournament selection	97
Translation		
	Translation	98
Trap -	Fron	200
Umda	Ггар	200
	Jnivariate marginal distribution algorithm	202
UniformCr		
	Jniform crossover	204
WalshExp		
	Valsh expansion   2     encion1	205
WalshExp	Walsh expansion of degree 1	207
WalshExp		
·	Walsh expansion of degree 2	209
	WalshTransformTerm	
\	Nalsh transform term	211

### **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.

    void bm_row_echelon_form (bit_matrix_t &A)

      Compute a row echelon form of a matrix.

    std::size_t bm_rank (const bit_matrix_t &A)

      Compute the rank of a matrix.

    bool bm_solve (bit_matrix_t &A, bit_vector_t &b)

      Solve a linear system.

    bool bm_solve_upper_triangular (bit_matrix_t &A, bit_vector_t &b)

      Solve a linear system in upper triangular form.

    bool bm_invert (bit_matrix_t &M, bit_matrix_t &N)

      Invert a bit matrix.

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

      Multiply a bit matrix and a bit vector.

    void bm transpose (const bit matrix t &M, bit matrix t &N)

      Transpose.
```

#### Types and functions related to bit

```
· typedef char bit_t

    bit_t bit_flip (bit_t b)

       Flip bit.
```

### Types and functions related to bit vectors

```
typedef std::vector< bit_t > bit_vector_t

    typedef std::pair< bit_vector_t, double > point_value_t

      Type to represent point value pairs.

    void bv_display (const bit_vector_t &v, std::ostream &stream)

      Display bit vector.

    bool bv_is_valid (const bit_vector_t &x)

      Check whether the bit vector is valid.

    bool bv_is_zero (const bit_vector_t &x)

      Check whether the bit vector is zero.

    int bv_hamming_weight (const bit_vector_t &x)

     Hamming weight.

    int bv_hamming_weight (const std::vector< bool > &x)

      Hamming weight.

    int bv_hamming_distance (const bit_vector_t &x, const bit_vector_t &y)

     Hamming distance between two bit vectors.

    bit_t bv_dot_product (const bit_vector_t &x, const bit_vector_t &y)

      Dot product.

    bit_t bv_dot_product (const bit_vector_t &x, const std::vector< bool > &y)

     Dot product.

    void bv_clear (bit_vector_t &x)

      Clear bit vector.

    void bv_flip (bit_vector_t &x, std::size_t i)

      Flip a single bit.

    void bv_flip (bit_vector_t &x, const bit_vector_t &mask)

      Flip many bits.

    void bv_random (bit_vector_t &x)

      Sample a random bit vector.

    void bv_random (bit_vector_t &x, int k)

      Sample a random bit vector with given Hamming weight.

    void bv_add (const bit_vector_t &src, bit_vector_t &dest)

      Add two bit vectors.

    void bv_add (const bit_vector_t &x, const bit_vector_t &y, bit_vector_t &dest)

      Add two bit vectors.

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

      Convert a bit vector to a bool vector.

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

      Convert a bool vector to a bit vector.

    std::size_t bv_to_size_type (const bit_vector_t &x)

      Convert a bit vector to a size_t.

    void bv_from_size_type (bit_vector_t &x, std::size_t index)

      Convert a size_t to a bit vector.
```

#### Types and functions related to permutations

```
    typedef std::vector< std::size_t > permutation_t
    Permutation type.
```

• bool perm\_is\_valid (const permutation\_t &permutation)

Check that a vector represents a permutation.

void perm\_random (permutation\_t &s)

Sample a random permutation.

#### Types and functions related to sparse bit matrices

```
    typedef std::vector< sparse_bit_vector_t > sparse_bit_matrix_t
    Sparse bit matrix.
```

void sbm\_display (const sparse\_bit\_matrix\_t &sbm, std::ostream &stream)

Display sparse bit matrix.

void bm\_to\_sbm (const bit\_matrix\_t &bm, sparse\_bit\_matrix\_t &sbm)

Convert a bit matrix to a sparse bit matrix.

void sbm\_multiply (const sparse\_bit\_matrix\_t &M, const bit\_vector\_t &x, bit\_vector\_t &y)

Multiply a sparse bit matrix and a bit vector.

#### Types and functions related to sparse bit vectors

```
    typedef std::vector< std::size_t > sparse_bit_vector_t
    Sparse bit vector.
```

void bv\_flip (bit\_vector\_t &x, const sparse\_bit\_vector\_t &sbv)
 Flip many bits.

void sbv\_display (const sparse\_bit\_vector\_t &v, std::ostream &stream)

Display sparse bit vector.

void bv\_to\_sbv (const bit\_vector\_t &bv, sparse\_bit\_vector\_t &sbv)

Convert a bit vector to a sparse bit vector.

#### 4.1.1 Detailed Description

top-level HNCO namespace

#### 4.1.2 Typedef Documentation

#### 4.1.2.1 bit\_t

typedef char bit\_t

Bit.

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

Definition at line 52 of file bit-vector.hh.

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

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

Sparse bit matrix.

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

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

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

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

Sparse bit vector.

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

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

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

#### 4.1.3 Function Documentation

#### 4.1.3.1 bm\_add\_rows()

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 ( \label{eq:bit_matrix_t & $M$} \ )
```

Set the matrix to the identity matrix.

#### Precondition

```
bm_is_square(M)
```

Definition at line 29 of file bit-matrix.cc.

#### 4.1.3.3 bm\_invert()

```
bool bm_invert ( \label{eq:bit_matrix_t & M, bit_matrix_t & N}  bit_matrix_t & N )
```

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 200 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 242 of file bit-matrix.cc.

#### 4.1.3.5 bm\_rank()

Compute the rank of a matrix.

Precondition

A must be in row echelon form.

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

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

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

Compute a row echelon form of a matrix.

Warning

A is modified by the function.

Definition at line 103 of file bit-matrix.cc.

#### 4.1.3.7 bm\_solve()

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

Solve a linear system.

Solve the linear equation Ax = b.

#### **Parameters**

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 150 of file bit-matrix.cc.

#### 4.1.3.8 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 181 of file bit-matrix.cc.

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

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

Convert a bool vector to a bit vector.

Warning

Vectors must be of the same size.

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

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

Convert a bit vector to a bool vector.

Warning

Vectors must be of the same size.

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

### 4.1.3.11 sbm\_multiply()

Multiply a sparse bit matrix and a bit vector.

The result is y = Mx.

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

# 4.2 hnco::algorithm Namespace Reference

Algorithms.

# **Namespaces**

• bm pbil

Boltzmann machine PBIL.

• hea

Herding evolutionary algorithm.

### Classes

· class Algorithm

Abstract search algorithm.

· class BiasedCrossover

Biased crossover.

· class CompactGa

Compact genetic algorithm.

• class CompleteSearch

Complete search.

· class Crossover

Crossover.

· class FirstAscentHillClimbing

First ascent hill climbing.

class GeneticAlgorithm

Genetic algorithm.

· class IterativeAlgorithm

Iterative search.

• class Mmas

Max-min ant system.

• class MuCommaLambdaEa

(mu, lambda) EA.

· class MuPlusLambdaEa

(mu+lambda) EA.

• class NpsPbil

Population-based incremental learning with negative and positive selection.

• class OnePlusLambdaCommaLambdaGa

(1+(lambda, lambda)) genetic algorithm.

· class OnePlusOneEa

(1+1) EA.

class Pbil

Population-based incremental learning.

class Population

Population.

class PvAlgorithm

Probability vector algorithm.

· class RandomLocalSearch

Random local search.

class RandomSearch

Random search.

· class Restart

Restart.

· class SimulatedAnnealing

Simulated annealing.

· class SteepestAscentHillClimbing

Steepest ascent hill climbing.

· class TournamentSelection

Population with tournament selection.

· class Umda

Univariate marginal distribution algorithm.

class UniformCrossover

Uniform crossover.

### **Functions**

```
template < class T > bool matrix_is_symmetric (const std::vector < std::vector < T > > &A)
```

Check for symmetric matrix.

template<class T >

bool matrix\_has\_diagonal (const std::vector< std::vector< T >> &A, T x)

Check for diagonal elements.

template<class T >

bool matrix has range (const std::vector< std::vector< T >> &A, T inf, T sup)

Check for element range.

template < class T >

bool matrix has dominant diagonal (const std::vector< std::vector< T >> &A)

Check for element range.

template < class T >

T square (T x)

Generic square function.

• double logistic (double x)

Logistic function (sigmoid)

# Type and functions related to probability vectors

```
    typedef std::vector< double > pv_t
```

Probability vector type.

double pv\_entropy (const pv\_t &pv)

Entropy of a probability vector.

void pv\_sample (const pv\_t &pv, bit\_vector\_t &x)

Sample a bit vector.

• void pv\_uniform (pv\_t &pv)

Probability vector of the uniform distribution.

void pv\_init (pv\_t &pv)

Initialize.

void pv add (pv t &pv, const bit vector t &x)

Accumulate a bit vector.

void pv\_add (pv\_t &pv, const bit\_vector\_t &x, double weight)

Accumulate a bit vector.

void pv\_average (pv\_t &pv, int count)

Average

void pv\_update (pv\_t &pv, double rate, const bit\_vector\_t &x)

Update a probability vector toward a bit vector.

void pv\_update (pv\_t &pv, double rate, const std::vector< double > &x)

Update a probability vector toward a probability vector.

void pv\_update (pv\_t &pv, double rate, const std::vector< double > &x, const std::vector< double > &y)

Update a probability vector toward a probability vector and away from another one.

void pv\_bound (pv\_t &pv, double lower\_bound, double upper\_bound)

Bound the components of a probability vector.

# 4.2.1 Detailed Description

Algorithms.

# 4.3 hnco::algorithm::bm\_pbil Namespace Reference

Boltzmann machine PBIL.

### **Classes**

• class BmPbil

Boltzmann machine PBIL.

class Model

Model of a Boltzmann machine.

• class ModelParameters

Parameters of a Boltzmann machine.

# 4.3.1 Detailed Description

Boltzmann machine PBIL.

# 4.4 hnco::algorithm::hea Namespace Reference

Herding evolutionary algorithm.

### Classes

· class BinaryHerding

Herding with binary variables.

struct BinaryMoment

Moment for binary variables.

class Hea

Herding evolutionary algorithm.

class SpinHerding

Herding with spin variables.

• struct SpinMoment

Moment for spin variables.

# 4.4.1 Detailed Description

Herding evolutionary algorithm.

# 4.5 hnco::exception Namespace Reference

Exceptions.

### **Classes**

· class Error

Error.

class Exception

Basic exception.

class LastEvaluation

Last evaluation.

· class LocalMaximum

Local maximum.

class MaximumReached

Maximum reached.

• class PointValueException

Point-value exception.

class TargetReached

target reached

# 4.5.1 Detailed Description

Exceptions.

# 4.6 hnco::function Namespace Reference

Functions to be maximized.

### Classes

· class AdditiveGaussianNoise

Additive Gaussian Noise.

· class Cache

Cache.

· class CallCounter

Call counter.

class DeceptiveJump

Deceptive jump.

class EqualProducts

Equal products.

· class Factorization

Factorization.

class FourPeaks

Four Peaks.

class Function

Function.

• class FunctionController

Function controller.

• class FunctionDecorator

Function decorator.

• class FunctionMapComposition

Composition of a function and a map.

· class FunctionModifier

Function modifier.

· class FunctionPlugin

Function plugin.

class Hiff

Hierarchical if and only if.

· class Jump

Jump.

• class Labs

Low autocorrelation binary sequences.

class LeadingOnes

Leading ones.

class LinearFunction

Linear function.

· class LongPath

Long path.

· class MaxSat

MAX-SAT.

• class Needle

Needle in a haystack.

· class Negation

Negation.

class NkLandscape

NK landscape.

class OnBudgetFunction

CallCounter with a limited number of evaluations.

class OneMax

OneMax.

· class Plateau

Plateau.

class PriorNoise

Prior noise.

class ProgressTracker

ProgressTracker.

· class Qubo

Quadratic unconstrained binary optimization.

• class Ridge

Ridge.

· class SinusSummationCancellation

Summation cancellation with sinus.

class SixPeaks

Six Peaks.

• class StopOnMaximum

Stop on maximum.

class StopOnTarget

Stop on target.

• class SummationCancellation

Summation cancellation.

class Trap

Trap.

class WalshExpansion

Walsh expansion.

class WalshExpansion1

Walsh expansion of degree 1.

• class WalshExpansion2

Walsh expansion of degree 2.

### **Functions**

std::ostream & operator<< (std::ostream &stream, const ProgressTracker::Event &event)</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 HammingSphere

Hamming sphere.

· class HammingSphereIterator

Hamming sphere neighborhood iterator.

class MultiBitFlip

Multi bit flip.

· class Neighborhood

Neighborhood.

· class NeighborhoodIterator

Neighborhood iterator.

· class SingleBitFlip

One bit neighborhood.

• class SingleBitFlipIterator

Single bit flip neighborhood iterator.

# 4.7.1 Detailed Description

Neighborhoods for local search.

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

# 4.8 hnco::random Namespace Reference

Pseudo random numbers.

### **Classes**

• struct Random

Random numbers.

# 4.8.1 Detailed Description

Pseudo random numbers.

# **Chapter 5**

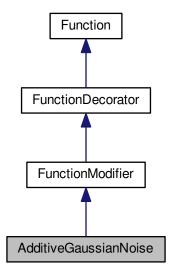
# **Class Documentation**

# 5.1 AdditiveGaussianNoise Class Reference

Additive Gaussian Noise.

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

Inheritance diagram for AdditiveGaussianNoise:



### **Public Member Functions**

- AdditiveGaussianNoise (Function \*function, double stddev)
   Constructor.
- double eval (const bit\_vector\_t &)

Evaluate a bit vector.

#### Information about the function

```
• size_t get_bv_size ()

Get bit vector size.
```

• double get\_maximum ()

Get the global maximum.

bool has\_known\_maximum ()

Check for a known maximum.

### **Private Attributes**

 std::normal\_distribution< double > \_dist Normal distribution.

### **Additional Inherited Members**

# 5.1.1 Detailed Description

Additive Gaussian Noise.

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

### 5.1.2 Member Function Documentation

```
5.1.2.1 get_maximum()
```

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

Get the global maximum.

**Exceptions** 

Error

Reimplemented from Function.

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

```
5.1.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

false

Reimplemented from Function.

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

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

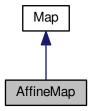
- · lib/hnco/functions/decorators/function-modifier.hh
- · lib/hnco/functions/decorators/function-modifier.cc

# 5.2 AffineMap Class Reference

Affine map.

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

Inheritance diagram for AffineMap:



# **Public Member Functions**

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

Random instance.

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

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)
    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.

Definition at line 257 of file map.hh.

### 5.2.2 Member Function Documentation

```
5.2.2.1 is_surjective()
bool is_surjective ( ) [virtual]
Check for surjective map.

Returns
    true if rank(_bm) == bm_num_rows(_bm)
Reimplemented from Map.
Definition at line 136 of file map.cc.
5.2.2.2 random()
void random (
```

int rows, int cols,

bool surjective )

Random instance.

### **Parameters**

rows	Number of rows
cols	Number of columns
surjective	Flag to ensure a surjective map

# **Exceptions**



Definition at line 99 of file map.cc.

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

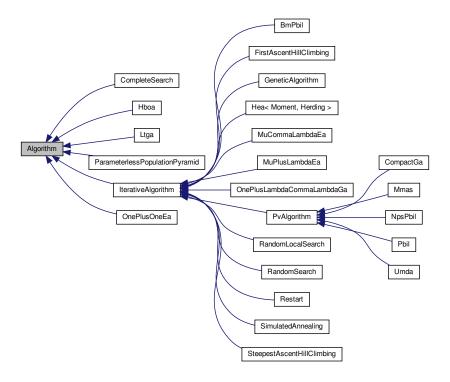
- · lib/hnco/map.hh
- · lib/hnco/map.cc

# 5.3 Algorithm Class Reference

Abstract search algorithm.

#include <hnco/algorithms/algorithm.hh>

Inheritance diagram for Algorithm:



### **Public Member Functions**

• Algorithm (int n)

Constructor.

virtual ∼Algorithm ()

Destructor.

### Optimization

· virtual void init ()

Initialization.

• virtual void maximize ()=0

Maximize.

#### Getters

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

Solution.

virtual size\_t get\_bv\_size ()

Get bit vector size.

#### **Setters**

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

Set function.

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

Set functions.

void set\_stream (std::ostream \*x)

Output stream.

### **Protected Member Functions**

• void random\_solution ()

Random solution.

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

Set solution.

void set\_solution (const bit\_vector\_t &x)

Set solution.

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

Update solution (strict)

void update\_solution (const point\_value\_t &pv)

Update solution (strict)

void update\_solution (const bit\_vector\_t &x)

Update solution (strict)

### **Protected Attributes**

• function::Function \* function

Function.

•  $std::vector < function::Function * > \_functions$ 

Functions.

• point\_value\_t \_solution

Solution.

### **Parameters**

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

# 5.3.1 Detailed Description

Abstract search algorithm.

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

Definition at line 38 of file algorithm.hh.

### 5.3.2 Member Data Documentation

### 5.3.2.1 \_functions

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

Functions.

Each thread has its own function.

Definition at line 49 of file algorithm.hh.

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

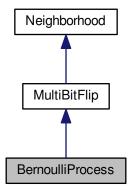
- · lib/hnco/algorithms/algorithm.hh
- · lib/hnco/algorithms/algorithm.cc

# 5.4 BernoulliProcess Class Reference

Bernoulli process.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for BernoulliProcess:



### **Public Member Functions**

```
• BernoulliProcess (int n)
```

Constructor.

• BernoulliProcess (int n, double p)

Constructor.

void set\_probability (double p)

Set probability.

### **Private Member Functions**

```
• void sample_bits ()
```

Sample bits.

void bernoulli\_process ()

Bernoulli process.

### **Private Attributes**

• std::bernoulli\_distribution \_bernoulli\_dist

Bernoulli distribution (biased coin)

 $\bullet \quad \mathsf{std} \text{::} \mathsf{binomial\_distribution} < \mathsf{int} > \_\mathsf{binomial\_dist}$ 

Binomial distribution.

bool <u>\_reservoir\_sampling</u> = false

Reservoir sampling.

### **Parameters**

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

Allow stay.

void set\_allow\_stay (bool x)

Set the flag \_allow\_stay.

### **Additional Inherited Members**

# 5.4.1 Detailed Description

### Bernoulli process.

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

Definition at line 220 of file neighborhood.hh.

### 5.4.2 Constructor & Destructor Documentation

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

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

Constructor.

#### **Parameters**

```
n Size of bit vectors
```

The Bernoulli probability is set to 1 / n.

Definition at line 255 of file neighborhood.hh.

### 5.4.2.2 BernoulliProcess() [2/2]

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

Constructor.

#### **Parameters**

n	Size of bit vectors
р	Bernoulli probability

Definition at line 265 of file neighborhood.hh.

### 5.4.3 Member Function Documentation

#### 5.4.3.1 set\_allow\_stay()

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

Set the flag \_allow\_stay.

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

Definition at line 292 of file neighborhood.hh.

### 5.4.3.2 set\_probability()

```
void set_probability ( \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 276 of file neighborhood.hh.

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

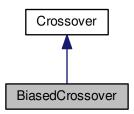
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 5.5 BiasedCrossover Class Reference

### Biased crossover.

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

Inheritance diagram for BiasedCrossover:



### **Public Member Functions**

• BiasedCrossover ()

Constructor.

- void breed (const bit\_vector\_t &parent1, const bit\_vector\_t &parent2, bit\_vector\_t &offspring)
   Breed.
- void set\_bias (double b)
   Set bias.

### **Private Attributes**

 std::bernoulli\_distribution \_bernoulli\_dist Bernoulli distribution.

### 5.5.1 Detailed Description

Biased crossover.

Definition at line 75 of file crossover.hh.

### 5.5.2 Member Function Documentation

# 5.5.2.1 breed()

### 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.

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

Sample a bit vector.

double error (const BinaryMoment &target)

Compute the error.

• double delta (const BinaryMoment &target)

Compute the norm of delta.

### **Setters**

• void set\_randomize\_bit\_order (bool x)

Randomize bit order.

• void set\_dynamics (int x)

Set the dynamics.

void set\_weight (double x)

Set the weight of second order moments.

### **Protected Member Functions**

void compute\_delta (const BinaryMoment &target)

Compute delta.

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

Sample a bit vector.

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

Sample a bit vector.

### **Protected Attributes**

• BinaryMoment \_count

Counter moment.

• BinaryMoment \_delta

Delta moment.

• permutation\_t \_permutation

Permutation.

• std::uniform\_int\_distribution< int > \_choose\_bit

Choose bit.

int \_time

Time.

#### **Parameters**

• bool randomize bit order = false

Randomize bit order.

• int \_dynamics = DYNAMICS\_MINIMIZE\_NORM

Dynamics.

double \_weight = 1

Weight of second order moments.

# 5.6.1 Detailed Description

Herding with binary variables.

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

# 5.6.2 Member Enumeration Documentation

### 5.6.2.1 anonymous enum

anonymous enum

#### Enumerator

DYNAMICS_MINIMIZE_NORM	Dynamics defined as minimization of a norm.	
DYNAMICS MAXIMIZE INNER PRODUCT	Dynamics defined as maximization of an inner product.	
	Generated by	, Doxygen

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

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

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

# 5.7 BinaryMoment Struct Reference

Moment for binary variables.

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

### **Public Member Functions**

```
• BinaryMoment (int n)
```

Constructor.

· void uniform ()

Set the moment to that of the uniform distribution.

• void init ()

Initialize.

void add (const bit\_vector\_t &x)

Accumulate a bit vector.

· void average (int count)

Compute average.

• void update (const BinaryMoment &p, double rate)

Update moment.

· void bound (double margin)

Bound moment.

double distance (const BinaryMoment &p) const

Distance.

• double norm\_2 () const

Compute the norm 2.

· double diameter () const

Compute the diameter.

· size\_t size () const

Size.

### **Public Attributes**

```
std::vector< std::vector< double >> _moment
```

Moment.

• double \_weight = 1

Weight of second order moments.

# 5.7.1 Detailed Description

Moment for binary variables.

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

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

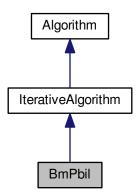
- · lib/hnco/algorithms/hea/moment-binary.hh
- lib/hnco/algorithms/hea/moment-binary.cc

# 5.8 BmPbil Class Reference

Boltzmann machine PBIL.

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

Inheritance diagram for BmPbil:



# **Public Types**

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

# **Public Member Functions**

• BmPbil (int n, int population\_size)

Constructor.

· void init ()

Initialization.

### **Private Member Functions**

· void iterate ()

Single iteration.

• void log ()

Log.

void sample (bit\_vector\_t &x)

Sample a bit vector.

• void sample\_asynchronous ()

Asynchronous sampling.

• void sample\_asynchronous\_full\_scan ()

Asynchronous sampling with full scan.

• void sample\_synchronous ()

Synchronous sampling.

#### **Private Attributes**

• log\_flags\_t\_log\_flags

Log flags.

Population \_population

Population.

· Model \_model

Model.

ModelParameters \_parameters\_all

Parameters averaged over all individuals.

• ModelParameters\_parameters\_best

Parameters averaged over selected individuals.

ModelParameters \_parameters\_worst

Parameters averaged over negatively selected individuals.

 $\bullet \quad \text{std::uniform\_int\_distribution} < \text{size\_t} > \_\text{choose\_bit} \\$ 

Uniform distribution on bit\_vector\_t components.

· permutation\_t \_permutation

Permutation.

### **Parameters**

• int \_selection\_size = 1

Selection size (number of selected individuals in the population)

• double \_rate = 1e-3

Learning rate.

• int \_num\_gs\_steps = 100

Number of gibbs sampler steps.

• int \_num\_gs\_cycles = 1

Number of gibbs sampler cycles.

• bool \_negative\_positive\_selection = false

Negative and positive selection.

• int sampling = SAMPLING ASYNCHRONOUS

Sampling mode.

int \_mc\_reset\_strategy = RESET\_NO\_RESET

MC reset strategy.

• void set\_selection\_size (int x)

Set the selection size.

• void set\_rate (double x)

Set the learning rate.

void set\_num\_gs\_steps (int x)

Set the number of gibbs sampler steps.

• void set\_num\_gs\_cycles (int x)

Set the number of gibbs sampler cycles.

• void set\_negative\_positive\_selection (bool x)

Set negative and positive selection.

void set\_sampling (int x)

Set the sampling mode.

void set\_mc\_reset\_strategy (int x)

Set the MC reset strategy.

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

Set log flags.

### **Additional Inherited Members**

# 5.8.1 Detailed Description

Boltzmann machine PBIL.

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

### Reference:

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

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

### 5.8.2 Member Enumeration Documentation

### 5.8.2.1 anonymous enum

anonymous enum

### Enumerator

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

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

### 5.8.2.2 anonymous enum

anonymous enum

### Enumerator

SAMPLING_ASYNCHRONOUS	Asynchronous sampling. A single component of the internal state is randomly selected then updated by Gibbs sampling. This step is repeated _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.

### 5.8.3 Member Function Documentation

### 5.8.3.1 set\_selection\_size()

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

Set the selection size.

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

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

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

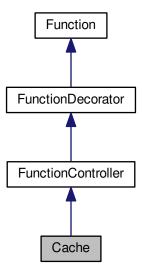
- lib/hnco/algorithms/bm-pbil/bm-pbil.hh
- lib/hnco/algorithms/bm-pbil/bm-pbil.cc

# 5.9 Cache Class Reference

### Cache.

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

Inheritance diagram for Cache:



### **Public Member Functions**

• Cache (Function \*function)

Constructor.

• bool provides\_incremental\_evaluation ()

Check whether the function provides incremental evaluation.

### **Evaluation**

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

5.9 Cache Class Reference 45

### **Private Attributes**

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

### **Additional Inherited Members**

# 5.9.1 Detailed Description

#### Cache.

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

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

Definition at line 355 of file function-controller.hh.

### 5.9.2 Constructor & Destructor Documentation

```
5.9.2.1 Cache()

Cache (

Function * function ) [inline]

Constructor.

Parameters

function | Decorated function |
```

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

### 5.9.3 Member Function Documentation

### 5.9.3.1 provides\_incremental\_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

Reimplemented from FunctionController.

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

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

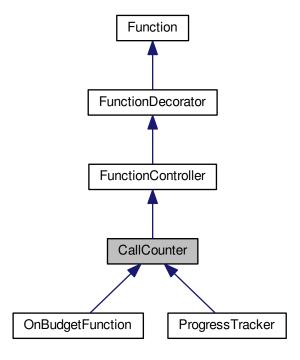
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 5.10 CallCounter Class Reference

Call counter.

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

Inheritance diagram for CallCounter:



### **Public Member Functions**

• CallCounter (Function \*function)

Constructor.

int get\_num\_calls ()

Get the number of calls.

#### **Evaluation**

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

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

Incremental evaluation.

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

Update after a safe evaluation.

#### **Protected Attributes**

int \_num\_calls

Number of calls.

# 5.10.1 Detailed Description

Call counter.

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

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

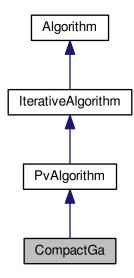
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 5.11 CompactGa Class Reference

Compact genetic algorithm.

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

Inheritance diagram for CompactGa:



# **Public Member Functions**

- CompactGa (int n)
  - Constructor.

Initialization.

void init ()

# 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:

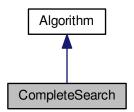
- · lib/hnco/algorithms/pv/compact-ga.hh
- lib/hnco/algorithms/pv/compact-ga.cc

# 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:

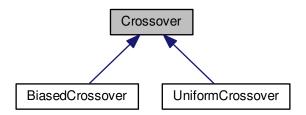
- lib/hnco/algorithms/complete-search.hh
- lib/hnco/algorithms/complete-search.cc

# 5.13 Crossover Class Reference

### Crossover.

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

Inheritance diagram for Crossover:



### **Public Member Functions**

virtual ∼Crossover ()

Destructor.

virtual void breed (const bit\_vector\_t &parent1, const bit\_vector\_t &parent2, bit\_vector\_t &offspring)=0
 Breed.

### 5.13.1 Detailed Description

# Crossover.

Definition at line 35 of file crossover.hh.

# 5.13.2 Member Function Documentation

### 5.13.2.1 breed()

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:

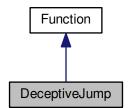
· lib/hnco/algorithms/ea/crossover.hh

# 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

```
_bv_size + _gap
```

Reimplemented from Function.

Definition at line 110 of file jump.hh.

### 5.14.2.2 has\_known\_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 106 of file jump.hh.

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

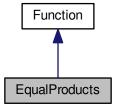
- lib/hnco/functions/jump.hh
- · lib/hnco/functions/jump.cc

# 5.15 EqualProducts Class Reference

# Equal products.

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

Inheritance diagram for EqualProducts:



# **Public Member Functions**

· EqualProducts ()

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

• void random (int n, double upper\_bound)

Random instance.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

### **Private Member Functions**

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

### **Private Attributes**

std::vector< double > \_numbers
 Numbers.

### **Friends**

· class boost::serialization::access

### 5.15.1 Detailed Description

### Equal products.

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

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

#### Reference:

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

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

### 5.15.2 Member Function Documentation

### 5.15.2.1 random()

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

### Random instance.

#### **Parameters**

n	Size of bit vector
upper_bound	Upper bound of numbers

5.16 Error Class Reference 55

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

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

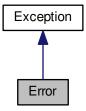
- · lib/hnco/functions/equal-products.hh
- · lib/hnco/functions/equal-products.cc

# 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:

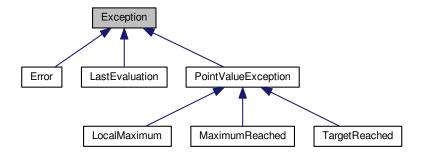
lib/hnco/functions/decorators/function-controller.hh

# 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:

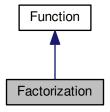
· lib/hnco/exception.hh

# 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:

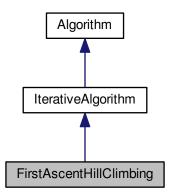
- · lib/hnco/functions/factorization.hh
- · lib/hnco/functions/factorization.cc

# 5.20 FirstAscentHillClimbing Class Reference

First ascent hill climbing.

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

Inheritance diagram for FirstAscentHillClimbing:



#### **Public Member Functions**

Constructor.

- FirstAscentHillClimbing (int n, neighborhood::NeighborhoodIterator \*neighborhood)
- void init ()

Random initialization.

void init (const bit\_vector\_t &x)

Explicit initialization.

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

Explicit initialization.

# **Protected Member Functions**

void iterate ()
 Single iteration.

# **Protected Attributes**

neighborhood::NeighborhoodIterator \* \_neighborhood
 Neighborhood.

# 5.20.1 Detailed Description

First ascent hill climbing.

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

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

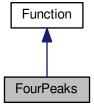
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/first-ascent-hill-climbing.cc

# 5.21 FourPeaks Class Reference

Four Peaks.

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

Inheritance diagram for FourPeaks:



### **Public Member Functions**

• FourPeaks (int bv\_size, int threshold)

Constructor.

size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit vector t &)

Evaluate a bit vector.

bool has\_known\_maximum ()

Check for a known maximum.

double get maximum ()

Get the global maximum.

#### **Private Attributes**

size\_t \_bv\_size

Bit vector size.

· int \_threshold

Threshold.

int maximum

Maximum.

### 5.21.1 Detailed Description

Four Peaks.

It is defined by

```
f(x) = \max\{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.

# 5.21.2 Member Function Documentation

### 5.21.2.1 get\_maximum()

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

Get the global maximum.

Returns

```
2 * _bv_size - _threshold - 1
```

Reimplemented from Function.

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

# 5.21.2.2 has\_known\_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

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

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

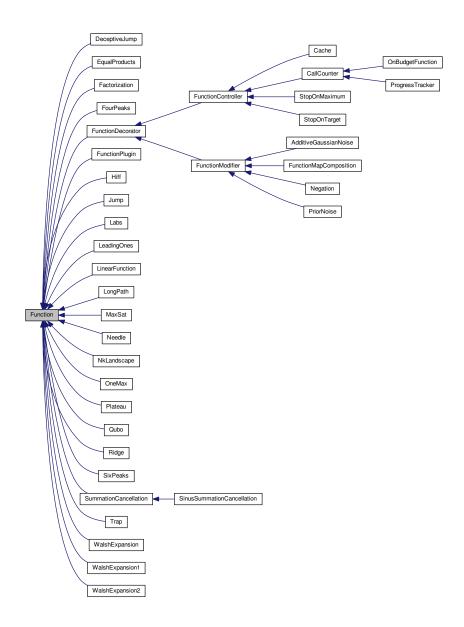
- lib/hnco/functions/four-peaks.hh
- lib/hnco/functions/four-peaks.cc

# 5.22 Function Class Reference

#### Function.

#include <hnco/functions/function.hh>

Inheritance diagram for Function:



# Classes

• struct WalshTransformTerm

Walsh transform term.

#### **Public Member Functions**

virtual ~Function ()
 Destructor.

#### Information about the function

virtual size\_t get\_bv\_size ()=0

Get bit vector size.

virtual double get\_maximum ()

Get the global maximum.

virtual bool has\_known\_maximum ()

Check for a known maximum.

· virtual bool provides incremental evaluation ()

Check whether the function provides incremental evaluation.

virtual void compute\_walsh\_transform (std::vector< Function::WalshTransformTerm > &terms)

Compute the Walsh transform of the function.

#### **Evaluation**

virtual double eval (const bit vector t &)=0

Evaluate a bit vector.

virtual double incremental\_eval (const bit\_vector\_t &x, double value, const hnco::sparse\_bit\_vector\_
 t &flipped\_bits)

Incremental evaluation.

virtual double safe\_eval (const bit\_vector\_t &x)

Safely evaluate a bit vector.

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

Update after a safe evaluation.

### **Display**

· virtual void display (std::ostream &stream)

Display

virtual void describe (const bit vector t &x, std::ostream &stream)

Describe a bit vector.

### 5.22.1 Detailed Description

#### Function.

Definition at line 39 of file function.hh.

#### 5.22.2 Member Function Documentation

#### 5.22.2.1 compute\_walsh\_transform()

Compute the Walsh transform of the function.

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

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

#### **Parameters**

terms | Vector of non zero terms of the Walsh transform

### Warning

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

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

Definition at line 33 of file function.cc.

#### 5.22.2.2 get\_maximum()

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

Get the global maximum.

### **Exceptions**

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 78 of file function.hh.

### 5.22.2.3 incremental\_eval()

Incremental evaluation.

### **Exceptions**

Error

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

Definition at line 131 of file function.hh.

#### 5.22.2.4 provides\_incremental\_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

false

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

Definition at line 86 of file function.hh.

### 5.22.2.5 safe\_eval()

Safely evaluate a bit vector.

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

Reimplemented in FunctionController.

Definition at line 141 of file function.hh.

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

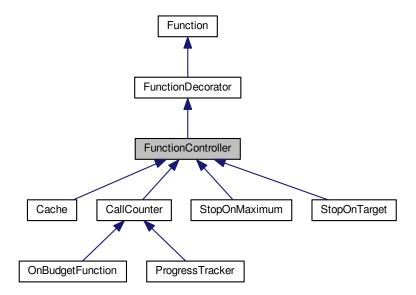
- lib/hnco/functions/function.hh
- · lib/hnco/functions/function.cc

# 5.23 FunctionController Class Reference

#### Function controller.

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

Inheritance diagram for FunctionController:



### **Public Member Functions**

• FunctionController (Function \*function)

Constructor.

### Information about the function

- size\_t get\_bv\_size ()
  - Get bit vector size.
- double get\_maximum ()

Get the global maximum.

- bool has\_known\_maximum ()
  - Check for a known maximum.
- bool provides\_incremental\_evaluation ()

Check whether the function provides incremental evaluation.

### **Evaluation**

double safe\_eval (const bit\_vector\_t &x)
 Safely evaluate a bit vector.

### **Display**

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

# **Additional Inherited Members**

### 5.23.1 Detailed Description

Function controller.

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

#### 5.23.2 Member Function Documentation

### 5.23.2.1 provides\_incremental\_evaluation()

bool provides\_incremental\_evaluation ( ) [inline], [virtual]

Check whether the function provides incremental evaluation.

Returns

true if the decorated function does

Reimplemented from Function.

Reimplemented in Cache.

Definition at line 63 of file function-controller.hh.

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

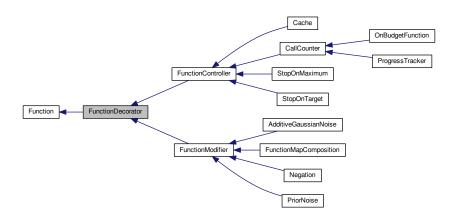
• lib/hnco/functions/decorators/function-controller.hh

# 5.24 FunctionDecorator Class Reference

Function decorator.

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

Inheritance diagram for FunctionDecorator:



# **Public Member Functions**

• FunctionDecorator (Function \*function) Constructor.

### **Protected Attributes**

• Function \* \_function Decorated function.

# 5.24.1 Detailed Description

Function decorator.

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

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

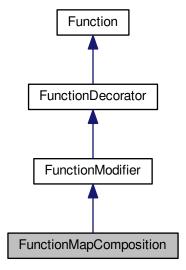
• lib/hnco/functions/decorators/function-decorator.hh

# 5.25 FunctionMapComposition Class Reference

Composition of a function and a map.

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

Inheritance diagram for FunctionMapComposition:



### **Public Member Functions**

```
• FunctionMapComposition (Function *function, Map *map)
```

Constructor.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

#### Information about the function

```
    size_t get_bv_size ()
        Get bit vector size.
    double get_maximum ()
        Get the global maximum.
    bool has_known_maximum ()
        Check for a known maximum.
```

### **Private Attributes**

```
Map * _mapMap.bit_vector_t _bv
```

### **Additional Inherited Members**

# 5.25.1 Detailed Description

Composition of a function and a map.

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

### 5.25.2 Constructor & Destructor Documentation

### 5.25.2.1 FunctionMapComposition()

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

Constructor.

# Precondition

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

# **Exceptions**

Error

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

### 5.25.3 Member Function Documentation

# 5.25.3.1 get\_maximum()

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

Get the global maximum.

#### **Exceptions**

Error

Reimplemented from Function.

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

### 5.25.3.2 has\_known\_maximum()

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

Check for a known maximum.

### Returns

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

Reimplemented from Function.

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

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

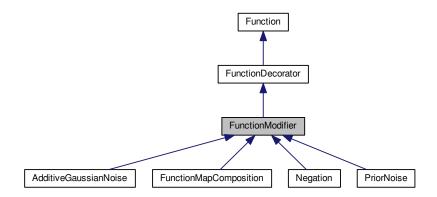
- · lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

# 5.26 FunctionModifier Class Reference

Function modifier.

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

Inheritance diagram for FunctionModifier:



# **Public Member Functions**

FunctionModifier (Function \*function)
 Constructor.

**Additional Inherited Members** 

# 5.26.1 Detailed Description

Function modifier.

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

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

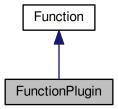
• lib/hnco/functions/decorators/function-modifier.hh

# 5.27 FunctionPlugin Class Reference

Function plugin.

#include <hnco/functions/plugin.hh>

Inheritance diagram for FunctionPlugin:



### **Public Member Functions**

• FunctionPlugin (int bv\_size, std::string path, std::string name)

Constructor.

• ∼FunctionPlugin ()

Destructor.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

# **Private Types**

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

#### **Private Attributes**

• size\_t \_bv\_size

Bit vector size.

void \* \_handle

Handle returned by dlopen.

extern\_function\_t \_extern\_function

Extern function.

# 5.27.1 Detailed Description

Function plugin.

Definition at line 34 of file plugin.hh.

# 5.27.2 Constructor & Destructor Documentation

# 5.27.2.1 FunctionPlugin()

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:

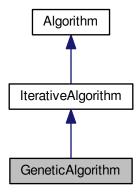
- lib/hnco/functions/plugin.hh
- lib/hnco/functions/plugin.cc

# 5.28 GeneticAlgorithm Class Reference

Genetic algorithm.

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

Inheritance diagram for GeneticAlgorithm:



# **Public Member Functions**

• GeneticAlgorithm (int n, int mu)

Constructor.

· void init ()

Initialization.

#### Setters

• void set\_mutation\_probability (double x)

Set the mutation probability.

• void set\_crossover\_probability (double x)

Set the crossover probability.

void set\_tournament\_size (int x)

Set the tournament size.

void set\_allow\_stay (bool x)

Set the flag \_allow\_stay.

### **Private Member Functions**

· void iterate ()

Single iteration.

# **Private Attributes**

TournamentSelection \_parents

Parents.

• TournamentSelection \_offsprings

Offsprings

• neighborhood::BernoulliProcess \_mutation

Mutation operator.

• std::bernoulli\_distribution \_do\_crossover

Do crossover.

• UniformCrossover \_crossover

Uniform crossover.

#### **Parameters**

```
• double _mutation_probability 
Mutation probability.
```

• double \_crossover\_probability = 0.5

Crossover probability.

• int \_tournament\_size = 10

Tournament size.

 bool <u>\_allow\_stay</u> = false Allow stay.

#### **Additional Inherited Members**

# 5.28.1 Detailed Description

Genetic algorithm.

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

#### Reference:

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

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

### 5.28.2 Constructor & Destructor Documentation

### 5.28.2.1 GeneticAlgorithm()

Constructor.

#### **Parameters**

n	Size of bit vectors
mu Population size	

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

#### 5.28.3 Member Function Documentation

### 5.28.3.1 set\_allow\_stay()

```
void set_allow_stay ( bool \ x \ ) \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 genetic-algorithm.hh.

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

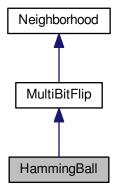
- · lib/hnco/algorithms/ea/genetic-algorithm.hh
- lib/hnco/algorithms/ea/genetic-algorithm.cc

# 5.29 HammingBall Class Reference

Hamming ball.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for HammingBall:



# **Public Member Functions**

HammingBall (int n, int r)
 Constructor.

# **Private Member Functions**

• void sample\_bits ()

Sample bits.

# **Private Attributes**

• int \_radius

Radius of the ball.

std::uniform\_int\_distribution< int > \_choose\_k
 Choose the distance to the center.

#### **Additional Inherited Members**

# 5.29.1 Detailed Description

Hamming ball.

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

Definition at line 304 of file neighborhood.hh.

### 5.29.2 Constructor & Destructor Documentation

### 5.29.2.1 HammingBall()

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

Constructor.

#### **Parameters**

n	Size of bit vectors
r	Radius of the ball

Definition at line 323 of file neighborhood.hh.

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

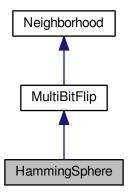
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 5.30 HammingSphere Class Reference

# Hamming sphere.

#include <hnco/neighborhoods/neighborhood.hh>

Inheritance diagram for HammingSphere:



# **Public Member Functions**

• HammingSphere (int n, int r)

Constructor.

void set\_radius (int r)

Set radius.

# **Private Member Functions**

void sample\_bits ()
 Sample bits.

### **Private Attributes**

• int \_radius

Radius of the sphere.

# **Additional Inherited Members**

# 5.30.1 Detailed Description

Hamming sphere.

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

Definition at line 341 of file neighborhood.hh.

### 5.30.2 Constructor & Destructor Documentation

### 5.30.2.1 HammingSphere()

```
\label{eq:hammingSphere} \begin{array}{cccc} \text{int } n, \\ & \text{int } r \;) & [\text{inline}] \end{array}
```

Constructor.

#### **Parameters**

n	Size of bit vectors
r	Radius of the sphere

Definition at line 357 of file neighborhood.hh.

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

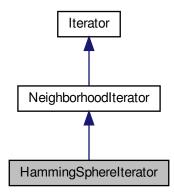
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

# 5.31 HammingSpherelterator Class Reference

Hamming sphere neighborhood iterator.

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

Inheritance diagram for HammingSphereIterator:



### **Public Member Functions**

• HammingSphereIterator (int n, int r)

Constructor.

· bool has\_next ()

Has next bit vector.

const bit\_vector\_t & next ()

Next bit vector.

### **Private Attributes**

bit\_vector\_t \_mask

Mutation mask.

· int radius

Radius of the ball.

· int \_index

Index of the next bit to shift to the right.

· int weight

Partial Hamming weight.

#### **Additional Inherited Members**

# 5.31.1 Detailed Description

Hamming sphere neighborhood iterator.

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

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

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

# 5.31.2 Constructor & Destructor Documentation

# 5.31.2.1 HammingSphereIterator()

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

Constructor.

### **Parameters**

n	Size of bit vectors
r	Radius of Hamming Ball

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

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

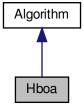
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

# 5.32 Hboa Class Reference

Hierarchical Bayesian Optimization Algorithm.

```
#include <hnco/algorithms/fast-efficient-p3/hboa.hh>
```

Inheritance diagram for Hboa:



### **Public Member Functions**

• Hboa (int n)

Constructor.

• void maximize ()

Maximize.

void set\_population\_size (int n)

Set population size.

#### **Private Attributes**

int \_population\_size = 10
 Population size.

### **Additional Inherited Members**

#### 5.32.1 Detailed Description

Hierarchical Bayesian Optimization Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 42 of file hboa.hh.

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

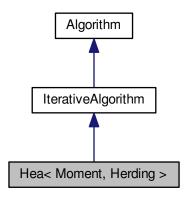
- · lib/hnco/algorithms/fast-efficient-p3/hboa.hh
- lib/hnco/algorithms/fast-efficient-p3/hboa.cc

# 5.33 Hea < Moment, Herding > Class Template Reference

Herding evolutionary algorithm.

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

Inheritance diagram for Hea< Moment, Herding >:



# **Public Types**

```
enum { RATE_CONSTANT, RATE_EXPONENTIAL, RATE_INVERSE }
```

• enum {

 ${\tt LOG\_ERROR, LOG\_DTU, LOG\_DELTA, LOG\_SELECTION,}$ 

LAST\_LOG }

typedef std::bitset < LAST\_LOG > log\_flags\_t

Type for log flags.

### **Public Member Functions**

• Hea (int n, int population\_size)

Constructor.

· void init ()

Initialization.

### Setters

void set\_herding (Herding \*x)

Set the herding algorithm.

• void set\_margin (double x)

Set the moment margin.

void set\_selection\_size (int x)

Set the selection size.

void set\_rate\_strategy (int x)

Set the rate strategy.

void set\_reset\_period (int x)

Set the reset period.

void set\_delay (int x)

Set the delay.

• void set\_initial\_rate (double x)

Set the initial value of the learning rate.

void set\_time\_constant (double x)

```
Set the time constant.
```

void set\_bound\_moment (bool x)

Set the bound moment after update.

void set\_weight (double weight)

Set weight.

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

Set log flags.

### **Private Member Functions**

• void iterate ()

Single iteration.

• void log ()

Log.

### **Private Attributes**

Moment target

Moment.

• Moment \_selection

Moment of selected individuals.

Moment uniform

Uniform moment.

• algorithm::Population \_population

Population.

Herding \* \_herding

Herding.

· double \_error\_cache

Error cache.

double \_dtu\_cache

Distance to uniform cache.

double \_delta\_cache

Delta cache.

· double selection cache

Selection distance cache.

log\_flags\_t \_log\_flags

Log flags.

#### **Parameters**

· double margin

Moment margin.

• int selection size = 1

Selection size.

int \_rate\_strategy = RATE\_CONSTANT

Rate strategy.

int \_reset\_period = 0

Reset period.

int \_delay = 10000

Delay.

double \_initial\_rate = 1e-4

Initial value of the learning rate.

• double \_time\_constant = 1000

Time constant.

• bool \_bound\_moment = false

Bound moment after update.

# **Additional Inherited Members**

# 5.33.1 Detailed Description

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.

# 5.33.2 Member Enumeration Documentation

### 5.33.2.1 anonymous enum

anonymous enum

#### Enumerator

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

Definition at line 54 of file hea.hh.

### 5.33.2.2 anonymous enum

anonymous enum

#### Enumerator

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

Definition at line 65 of file hea.hh.

# 5.33.3 Constructor & Destructor Documentation

Constructor.

**Parameters** 

```
n Size of bit vectors
```

\_margin is initialized to 1 / n.

Definition at line 234 of file hea.hh.

# 5.33.4 Member Function Documentation

# 5.33.4.1 set\_reset\_period()

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

Set the reset period.

**Parameters** 

```
x Reset period
```

 $x \le 0$  means no reset.

Definition at line 281 of file hea.hh.

# 5.33.4.2 set\_selection\_size()

Set the selection size.

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

Definition at line 270 of file hea.hh.

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

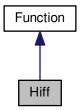
• lib/hnco/algorithms/hea/hea.hh

# 5.34 Hiff Class Reference

Hierarchical if and only if.

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

Inheritance diagram for Hiff:



# **Public Member Functions**

• Hiff (int bv\_size)

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

double get\_maximum ()

Get the global maximum.

# **Private Attributes**

• size\_t \_bv\_size

Bit vector size.

size\_t \_depth

Tree depth.

5.34 Hiff Class Reference 89

# 5.34.1 Detailed Description

Hierarchical if and only if.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 165 of file theory.hh.

### 5.34.2 Member Function Documentation

```
5.34.2.1 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.

### 5.34.2.2 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

# 5.35 HncoEvaluator Class Reference

Evaluator for HNCO functions.

#include <hnco/algorithms/fast-efficient-p3/hnco-evaluator.hh>

Inheritance diagram for HncoEvaluator:



### **Public Member Functions**

• HncoEvaluator (hnco::function::Function \*function)

Constructor.

float evaluate (const std::vector< bool > &x)

Evaluate a bit vector.

# **Private Attributes**

- hnco::function::Function \* \_function
   HNCO function.
- hnco::bit\_vector\_t \_bv

Argument of HNCO function.

# 5.35.1 Detailed Description

Evaluator for HNCO functions.

Definition at line 34 of file hnco-evaluator.hh.

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

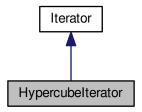
· lib/hnco/algorithms/fast-efficient-p3/hnco-evaluator.hh

# 5.36 Hypercubelterator Class Reference

Hypercube iterator.

#include <hnco/iterator.hh>

Inheritance diagram for Hypercubelterator:



### **Public Member Functions**

• Hypercubelterator (int n)

Constructor.

bool has\_next ()

Has next bit vector.

• const bit\_vector\_t & next ()

Next bit vector.

### **Additional Inherited Members**

## 5.36.1 Detailed Description

Hypercube iterator.

Implemented as a simple binary adder.

Definition at line 69 of file iterator.hh.

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

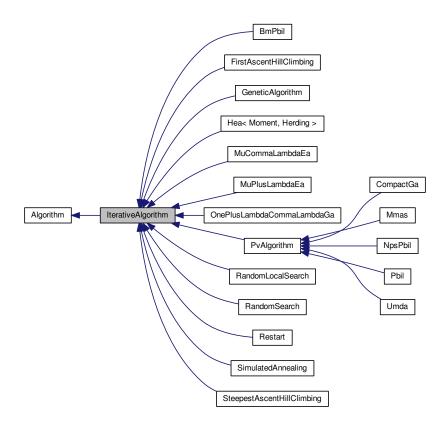
- · lib/hnco/iterator.hh
- lib/hnco/iterator.cc

# 5.37 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.

### Setters

void set\_num\_iterations (int x)
 Set the number of iterations.

## **Protected Member Functions**

- virtual void iterate ()=0
   Single iteration.
- virtual void log ()

Log.

## **Protected Attributes**

```
• int _iteration
```

Current iteration.

• bool \_something\_to\_log Something to log.

### **Parameters**

```
• int _num_iterations = 0 
Number of iterations.
```

## 5.37.1 Detailed Description

Iterative search.

Definition at line 139 of file algorithm.hh.

## 5.37.2 Constructor & Destructor Documentation

## 5.37.2.1 IterativeAlgorithm()

Constructor.

**Parameters** 

```
n Size of bit vectors
```

Definition at line 169 of file algorithm.hh.

### 5.37.3 Member Function Documentation

### 5.37.3.1 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.

### 5.37.3.2 set\_num\_iterations()

Set the number of iterations.

#### **Parameters**

```
x Number of iterations
```

 $x \le 0$  means indefinite

Definition at line 193 of file algorithm.hh.

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

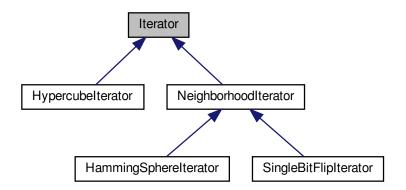
- lib/hnco/algorithms/algorithm.hh
- lib/hnco/algorithms/algorithm.cc

## 5.38 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 void init ()

Initialization.

• virtual bool has\_next ()=0

Has next bit vector.

• virtual const bit\_vector\_t & next ()=0

Next bit vector.

### **Protected Attributes**

bit\_vector\_t \_current

Current bit vector.

• bool \_initial\_state = true

Flag for initial state.

## 5.38.1 Detailed Description

Iterator over bit vectors.

Definition at line 34 of file iterator.hh.

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

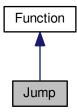
· lib/hnco/iterator.hh

## 5.39 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.

## 5.39.1 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.

5.40 Labs Class Reference 97

## 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 66 of file jump.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 62 of file jump.hh.

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

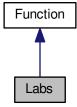
- · lib/hnco/functions/jump.hh
- lib/hnco/functions/jump.cc

## 5.40 Labs Class Reference

Low autocorrelation binary sequences.

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

Inheritance diagram for Labs:



### **Public Member Functions**

• Labs (int n)

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

### **Private Attributes**

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

## 5.40.1 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

## 5.41 LastEvaluation Class Reference

Last evaluation.

#include <hnco/exception.hh>

Inheritance diagram for LastEvaluation:



## 5.41.1 Detailed Description

Last evaluation.

Definition at line 79 of file exception.hh.

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

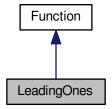
· lib/hnco/exception.hh

## 5.42 LeadingOnes Class Reference

Leading ones.

#include <hnco/functions/theory.hh>

Inheritance diagram for LeadingOnes:



## **Public Member Functions**

• LeadingOnes (int bv\_size)

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

• double get\_maximum ()

Get the global maximum.

## **Private Attributes**

size\_t \_bv\_size

Bit vector size.

## 5.42.1 Detailed Description

Leading ones.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 93 of file theory.hh.

### 5.42.2 Member Function Documentation

```
5.42.2.1 get_maximum()
```

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

Get the global maximum.

Returns

\_bv\_size

Reimplemented from Function.

Definition at line 117 of file theory.hh.

## 5.42.2.2 has\_known\_maximum()

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

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 113 of file theory.hh.

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

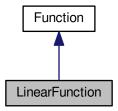
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.43 LinearFunction Class Reference

Linear function.

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

Inheritance diagram for LinearFunction:



## **Public Member Functions**

• LinearFunction ()

Constructor.

• void random (int n)

Random instance.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

• double get\_maximum ()

Get the global maximum.

### **Private Member Functions**

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

### **Private Attributes**

std::vector< double > \_weights Weights.

## **Friends**

· class boost::serialization::access

## 5.43.1 Detailed Description

Linear function.

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

## 5.43.2 Member Function Documentation

```
5.43.2.1 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.

## 5.43.2.2 random()

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

Random instance.

### **Parameters**

```
n Size of bit vectors
```

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

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

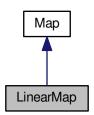
- · lib/hnco/functions/linear-function.hh
- lib/hnco/functions/linear-function.cc

# 5.44 LinearMap Class Reference

## Linear map.

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

Inheritance diagram for LinearMap:



### **Public Member Functions**

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

Random instance.

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

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)
 Load.

## **Private Attributes**

• bit\_matrix\_t \_bm

Bit matrix.

## **Friends**

· class boost::serialization::access

## 5.44.1 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 n x m bit matrix.

Definition at line 193 of file map.hh.

### 5.44.2 Member Function Documentation

```
5.44.2.1 is_surjective()
```

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

Check for surjective map.

### Returns

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

Reimplemented from Map.

Definition at line 90 of file map.cc.

## 5.44.2.2 random()

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

Random instance.

#### **Parameters**

rows	Number of rows
cols	Number of columns
surjective	Flag to ensure a surjective map

## **Exceptions**

Error

Definition at line 61 of file map.cc.

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

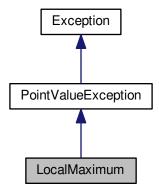
- · lib/hnco/map.hh
- · lib/hnco/map.cc

## 5.45 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**

## 5.45.1 Detailed Description

Local maximum.

Definition at line 70 of file exception.hh.

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

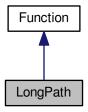
lib/hnco/exception.hh

# 5.46 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.

## 5.46.1 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:

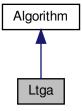
- · lib/hnco/functions/long-path.hh
- · lib/hnco/functions/long-path.cc

## 5.47 Ltga Class Reference

Linkage Tree Genetic Algorithm.

#include <hnco/algorithms/fast-efficient-p3/ltga.hh>

Inheritance diagram for Ltga:



### **Public Member Functions**

• Ltga (int n)

Constructor.

• void maximize ()

Maximize.

void set\_population\_size (int n)

Set population size.

### **Private Attributes**

• int \_population\_size = 10 Population size.

### **Additional Inherited Members**

### 5.47.1 Detailed Description

Linkage Tree Genetic Algorithm.

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

Author: Brian W. Goldman

Integrated into HNCO by Arnaud Berny

Definition at line 40 of file Itga.hh.

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

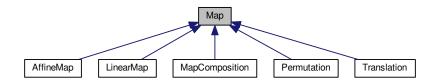
- · lib/hnco/algorithms/fast-efficient-p3/ltga.hh
- lib/hnco/algorithms/fast-efficient-p3/ltga.cc

## 5.48 Map Class Reference

### Map.

#include <hnco/map.hh>

Inheritance diagram for Map:



## **Public Member Functions**

virtual ∼Map ()

Destructor.

virtual void map (const bit\_vector\_t &input, bit\_vector\_t &output)=0
 Map.

virtual size\_t get\_input\_size ()=0

Get input size.

• virtual size\_t get\_output\_size ()=0

Get output size.

• virtual bool is\_surjective ()

Check for surjective map.

## 5.48.1 Detailed Description

Мар.

Definition at line 39 of file map.hh.

## 5.48.2 Member Function Documentation

```
5.48.2.1 is_surjective()
```

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

Check for surjective map.

Returns

false

Reimplemented in MapComposition, AffineMap, LinearMap, Permutation, and Translation.

Definition at line 59 of file map.hh.

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

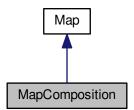
· lib/hnco/map.hh

# 5.49 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.

## 5.49.1 Detailed Description

Map composition.

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

Definition at line 327 of file map.hh.

## 5.49.2 Constructor & Destructor Documentation

# 5.49.2.1 MapComposition()

Constructor.

### **Parameters**

outer	outer map
inner	inner map

### Precondition

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

Definition at line 351 of file map.hh.

## 5.49.3 Member Function Documentation

## 5.49.3.1 is\_surjective()

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

Check for surjective map.

Returns

true if both maps are surjective

Reimplemented from Map.

Definition at line 375 of file map.hh.

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

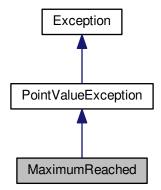
· lib/hnco/map.hh

## 5.50 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**

## 5.50.1 Detailed Description

Maximum reached.

Definition at line 52 of file exception.hh.

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

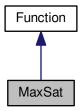
• lib/hnco/exception.hh

## 5.51 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.

## 5.51.1 Detailed Description

MAX-SAT.

Reference:

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

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

### 5.51.2 Member Function Documentation

```
5.51.2.1 load()
```

Load an instance.

**Exceptions** 

Error

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

```
5.51.2.2 random() [1/2]
```

Random instance.

### **Parameters**

n	Size of bit vectors
k	Number of literals per clause
C Gener	Number of clauses

Definition at line 38 of file max-sat.cc.

```
5.51.2.3 random() [2/2] void random ( const bit_vector_t & solution, int k, int c )
```

Random instance with satisfiable expression.

### Warning

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

### **Parameters**

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

Definition at line 66 of file max-sat.cc.

### 5.51.3 Member Data Documentation

```
5.51.3.1 _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

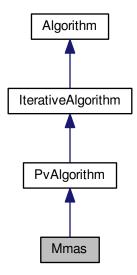
5.52 Mmas Class Reference 115

## 5.52 Mmas Class Reference

Max-min ant system.

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

Inheritance diagram for Mmas:



## **Public Member Functions**

- Mmas (int n)
  - Constructor.
- void init ()

Initialization.

## Setters

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

### **Protected Member Functions**

• void iterate ()

Single iteration.

### **Protected Attributes**

bit\_vector\_t \_x
 Candidate solution.

#### **Parameters**

```
    std::function< bool(double, double)> _compare = std::greater_equal<double>()
    Binary operator for comparing evaluations.
```

```
• double _rate = 1e-3 
Learning rate.
```

### **Additional Inherited Members**

### 5.52.1 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

## 5.53 Model Class Reference

Model of a Boltzmann machine.

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

### **Public Member Functions**

```
• Model (int n)
```

Constructor.

void init ()

Initialize.

· void reset\_mc ()

Reset Markov chain.

void gibbs\_sampler (size\_t i)

A Gibbs sampler cycle.

void gibbs\_sampler\_synchronous ()

A synchronous Gibbs sampler.

const bit\_vector\_t & get\_state ()

Get the state of the Gibbs sampler.

void update (const ModelParameters &p, const ModelParameters &q, double rate)

Update parameters in the direction of p and away from q.

• double norm\_infinite ()

Infinite norm of the parameters.

• double norm\_l1 ()

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.

## 5.53.1 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

## 5.54 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

## 5.54.1 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:

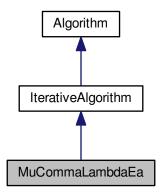
- · lib/hnco/algorithms/bm-pbil/model.hh
- lib/hnco/algorithms/bm-pbil/model.cc

## 5.55 MuCommaLambdaEa Class Reference

(mu, lambda) EA.

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

Inheritance diagram for MuCommaLambdaEa:



### **Public Member Functions**

• MuCommaLambdaEa (int n, int mu, int lambda)

Constructor.

· void init ()

Initialization.

### Setters

- void set\_mutation\_probability (double x)
  - Set the mutation probability.
- void set\_allow\_stay (bool x)

Set the flag \_allow\_stay.

### **Private Member Functions**

• void iterate ()

Single iteration.

### **Private Attributes**

• Population \_parents

Parents.

Population \_offsprings

Offsprings.

• neighborhood::BernoulliProcess \_mutation

Mutation operator.

std::uniform\_int\_distribution< int > \_select\_parent

Select parent.

#### **Parameters**

• double \_mutation\_probability Mutation probability.

• bool <u>\_allow\_stay</u> = false *Allow stay*.

### **Additional Inherited Members**

## 5.55.1 Detailed Description

(mu, lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

### 5.55.2 Constructor & Destructor Documentation

### 5.55.2.1 MuCommaLambdaEa()

```
MuCommaLambdaEa (
          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 79 of file mu-comma-lambda-ea.hh.

### 5.55.3 Member Function Documentation

### 5.55.3.1 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 102 of file mu-comma-lambda-ea.hh.

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

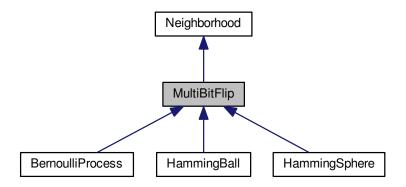
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-comma-lambda-ea.cc

# 5.56 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**

### 5.56.1 Detailed Description

Multi bit flip.

Definition at line 183 of file neighborhood.hh.

### 5.56.2 Constructor & Destructor Documentation

### 5.56.2.1 MultiBitFlip()

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

Constructor.

### **Parameters**

```
n Size of bit vectors
```

Definition at line 206 of file neighborhood.hh.

### 5.56.3 Member Function Documentation

## 5.56.3.1 bernoulli\_trials()

```
void bernoulli_trials ( \quad \text{int } k \text{ ) } \quad [\text{protected}]
```

Sample a given number of bits using Bernoulli trials.

#### **Parameters**

k Number of bits to sample

Definition at line 34 of file neighborhood.cc.

### 5.56.3.2 reservoir\_sampling()

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:

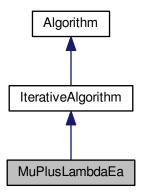
- lib/hnco/neighborhoods/neighborhood.hh
- lib/hnco/neighborhoods/neighborhood.cc

## 5.57 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**

## 5.57.1 Detailed Description

(mu+lambda) EA.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

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

## 5.57.2 Constructor & Destructor Documentation

## 5.57.2.1 MuPlusLambdaEa()

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.

## 5.57.3 Member Function Documentation

### 5.57.3.1 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:

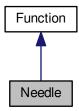
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.hh
- lib/hnco/algorithms/ea/mu-plus-lambda-ea.cc

## 5.58 Needle Class Reference

Needle in a haystack.

#include <hnco/functions/theory.hh>

Inheritance diagram for Needle:



### **Public Member Functions**

Needle (int bv\_size)

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

• double get\_maximum ()

Get the global maximum.

### **Private Attributes**

• size\_t \_bv\_size

Bit vector size.

## 5.58.1 Detailed Description

Needle in a haystack.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 129 of file theory.hh.

## 5.58.2 Member Function Documentation

```
5.58.2.1 get_maximum()
double get_maximum ( ) [inline], [virtual]
Get the global maximum.
Returns
     1
Reimplemented from Function.
Definition at line 153 of file theory.hh.
5.58.2.2 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.
```

· lib/hnco/functions/theory.hh

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

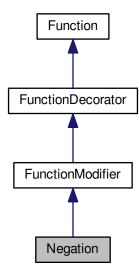
• lib/hnco/functions/theory.cc

# 5.59 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**

## 5.59.1 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.

## 5.59.2 Member Function Documentation

```
5.59.2.1 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.

```
5.59.2.2 has_known_maximum()
```

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

Check for a known maximum.

Returns

false

Reimplemented from Function.

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

### 5.59.2.3 provides\_incremental\_evaluation()

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

Check whether the function provides incremental evaluation.

Returns

true

Reimplemented from Function.

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

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

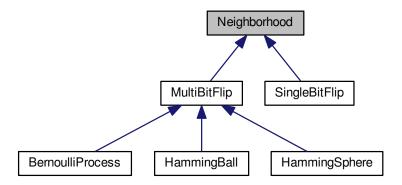
- · lib/hnco/functions/decorators/function-modifier.hh
- lib/hnco/functions/decorators/function-modifier.cc

# 5.60 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.

### 5.60.1 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.

### 5.60.2 Constructor & Destructor Documentation

### 5.60.2.1 Neighborhood()

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

Constructor.

### **Parameters**

```
n Size of bit vectors
```

Definition at line 86 of file neighborhood.hh.

### 5.60.3 Member Function Documentation

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.

### 5.60.3.2 mutate()

Mutate.

In-place mutation of the bit vector.

### **Parameters**

bv Bit vector to mu
---------------------

Definition at line 134 of file neighborhood.hh.

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

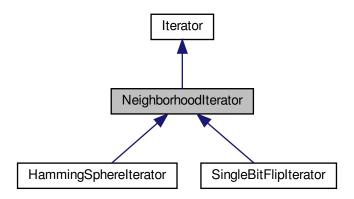
• lib/hnco/neighborhoods/neighborhood.hh

# 5.61 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**

## 5.61.1 Detailed Description

Neighborhood iterator.

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

### 5.61.2 Constructor & Destructor Documentation

### 5.61.2.1 NeighborhoodIterator()

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

Constructor.

### **Parameters**

```
n Size of bit vectors
```

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

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

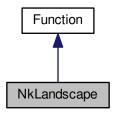
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

# 5.62 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

## 5.62.1 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.

## 5.62.2 Member Function Documentation

## 5.62.2.1 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 32 of file nk-landscape.cc.

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

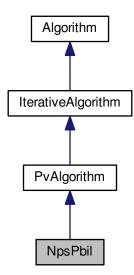
- lib/hnco/functions/nk-landscape.hh
- lib/hnco/functions/nk-landscape.cc

# 5.63 NpsPbil Class Reference

Population-based incremental learning with negative and positive selection.

#include <hnco/algorithms/pv/nps-pbil.hh>

Inheritance diagram for NpsPbil:



## **Public Member Functions**

- NpsPbil (int n, int population\_size)
   Constructor.
- void init ()

  Initialization.

## Setters

- void set\_selection\_size (int x)
  - Set the selection size.
- void set\_rate (double x)
   Set the learning rate.

## **Protected Member Functions**

• void iterate ()

Single iteration.

### **Protected Attributes**

- pv\_t \_mean\_best

Mean of best individuals.

pv\_t \_mean\_worst

Mean of worst individuals.

#### **Parameters**

Learning rate.

### **Additional Inherited Members**

### 5.63.1 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:

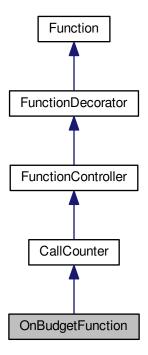
- lib/hnco/algorithms/pv/nps-pbil.hh
- lib/hnco/algorithms/pv/nps-pbil.cc

# 5.64 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** 

## 5.64.1 Detailed Description

CallCounter with a limited number of evaluations.

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

## 5.64.2 Member Function Documentation

Evaluate a bit vector.

**Exceptions** 

LastEvaluation

Reimplemented from CallCounter.

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

## 5.64.2.2 incremental\_eval()

Incremental evaluation.

**Exceptions** 

LastEvaluation

Reimplemented from CallCounter.

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

### 5.64.2.3 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:

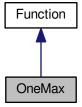
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

## 5.65 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.

### 5.65.1 Detailed Description

### OneMax.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 36 of file theory.hh.

### 5.65.2 Member Function Documentation

```
5.65.2.1 get_maximum()
```

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

Get the global maximum.

Returns

\_bv\_size

Reimplemented from Function.

Definition at line 57 of file theory.hh.

### 5.65.2.2 has\_known\_maximum()

bool has\_known\_maximum ( ) [inline], [virtual]

Check for a known maximum.

Returns

true

Reimplemented from Function.

Definition at line 61 of file theory.hh.

### 5.65.2.3 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:

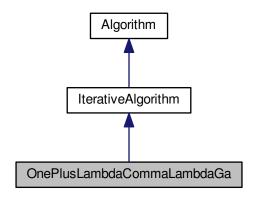
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

## 5.66 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**

### 5.66.1 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.

## 5.66.2 Constructor & Destructor Documentation

### 5.66.2.1 OnePlusLambdaCommaLambdaGa()

```
OnePlusLambdaCommaLambdaGa (
                int n,
                int lambda ) [inline]
```

Constructor.

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

### **Parameters**

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:

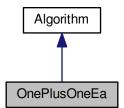
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.hh
- lib/hnco/algorithms/ea/one-plus-lambda-comma-lambda-ga.cc

## 5.67 OnePlusOneEa Class Reference

```
(1+1) EA.
```

#include <hnco/algorithms/ea/one-plus-one-ea.hh>

Inheritance diagram for OnePlusOneEa:



### **Public Member Functions**

• OnePlusOneEa (int n)

Constructor.

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

Set function.

· void init ()

Initialization.

· void maximize ()

Maximize.

#### 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**

## 5.67.1 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.

## 5.67.2 Constructor & Destructor Documentation

### 5.67.2.1 OnePlusOneEa()

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

Constructor.

**Parameters** 

```
n Size of bit vectors
```

\_mutation\_probability is initialized to 1 / n.

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

### 5.67.3 Member Function Documentation

## 5.67.3.1 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 126 of file one-plus-one-ea.hh.

## 5.67.3.2 set\_num\_iterations()

Set the number of iterations.

## **Parameters**

x Number of iterations

 $x \le 0$  means indefinite

Definition at line 116 of file one-plus-one-ea.hh.

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

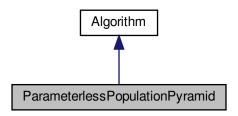
• lib/hnco/algorithms/ea/one-plus-one-ea.hh

# 5.68 ParameterlessPopulationPyramid Class Reference

Parameter-less Population Pyramid.

#include <hnco/algorithms/fast-efficient-p3/p3.hh>

Inheritance diagram for ParameterlessPopulationPyramid:



### **Public Member Functions**

• ParameterlessPopulationPyramid (int n)

Constructor.

· void maximize ()

Maximize.

### **Additional Inherited Members**

## 5.68.1 Detailed Description

Parameter-less Population Pyramid.

Implemention of the Parameter-less Population Pyramid (P3 for short).

Author: Brian W. Goldman

Reference:

"Fast and Efficient Black Box Optimization using the Parameter-less Population Pyramid" by B. W. Goldman and W. F. Punch

Integrated into HNCO by Arnaud Berny

Definition at line 44 of file p3.hh.

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

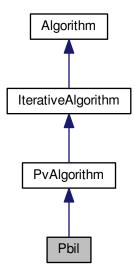
- lib/hnco/algorithms/fast-efficient-p3/p3.hh
- lib/hnco/algorithms/fast-efficient-p3/p3.cc

# 5.69 Pbil Class Reference

Population-based incremental learning.

#include <hnco/algorithms/pv/pbil.hh>

Inheritance diagram for Pbil:



## **Public Member Functions**

- Pbil (int n, int population\_size)

  Constructor.
- void init ()

Initialization.

## Setters

- void set\_selection\_size (int x)

  Set the selection size.
- void set\_rate (double x)

Set the learning rate.

## **Protected Member Functions**

• void iterate ()

Single iteration.

### **Protected Attributes**

• Population \_population

Population.

pv\_t \_mean

Mean of selected bit vectors.

#### **Parameters**

```
    int _selection_size = 1
        Selection size.
    double _rate = 1e-3
```

Learning rate.

### **Additional Inherited Members**

## 5.69.1 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:

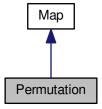
- · lib/hnco/algorithms/pv/pbil.hh
- lib/hnco/algorithms/pv/pbil.cc

## 5.70 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)
    Load.
```

## **Private Attributes**

• permutation\_t \_permutation Permutation.

## **Friends**

· class boost::serialization::access

### 5.70.1 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.

## 5.70.2 Member Function Documentation

```
5.70.2.1 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:

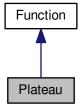
- · lib/hnco/map.hh
- · lib/hnco/map.cc

## 5.71 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.
```

## 5.71.1 Detailed Description

### Plateau.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 239 of file theory.hh.

### 5.71.2 Member Function Documentation

```
5.71.2.1 get_maximum()
```

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

Get the global maximum.

Returns

```
_bv_size + 2
```

Reimplemented from Function.

Definition at line 263 of file theory.hh.

### 5.71.2.2 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:

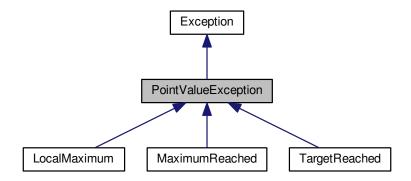
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

# 5.72 PointValueException Class Reference

Point-value exception.

#include <hnco/exception.hh>

Inheritance diagram for PointValueException:



### **Public Member Functions**

- PointValueException (const point\_value\_t &pv)
   Constructor.
- const point\_value\_t & get\_point\_value () const Get point-value.

### **Protected Attributes**

point\_value\_t \_pv
 Point-value.

### 5.72.1 Detailed Description

Point-value exception.

Definition at line 38 of file exception.hh.

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

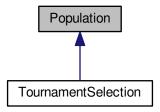
• lib/hnco/exception.hh

# 5.73 Population Class Reference

## Population.

#include <hnco/algorithms/population.hh>

Inheritance diagram for Population:



## **Public Types**

typedef std::pair< size\_t, double > index\_value\_t
 Index-value type.

## **Public Member Functions**

• Population (int population\_size, int n)

Constructor.

• std::size\_t size () const

Size.

• void random ()

Initialize the population with random bit vectors.

bit\_vector\_t & get\_bv (int i)

Get a bit vector.

• const bit\_vector\_t & get\_bv (int i) const

Get a bit vector.

## Get sorted bit vectors

• const bit\_vector\_t & get\_best\_bv (int i) const

Get best bit vector.

• const bit\_vector\_t & get\_best\_bv () const

Get best bit vector.

• const bit\_vector\_t & get\_worst\_bv (int i) const

Get worst bit vector.

## Get sorted values

```
    double get_best_value (int i) const
Get best value.
```

double get\_best\_value () const

Get best value.

### **Evaluation and sorting**

• void eval (function::Function \*function)

Evaluate the population.

void eval (const std::vector< function::Function \*> &functions)

Parallel evaluation of the population.

• void sort ()

Sort the lookup table.

#### Selection

• void plus\_selection (const Population &offsprings)

Plus selection.

void comma selection (const Population & offsprings)

Comma selection.

### **Protected Attributes**

```
std::vector< bit_vector_t > _bvs
```

Bit vectors.

std::vector< index\_value\_t > \_lookup

Lookup table.

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

## 5.73.1 Detailed Description

## Population.

Definition at line 36 of file population.hh.

### 5.73.2 Member Function Documentation

### 5.73.2.1 comma\_selection()

Comma selection.

### Precondition

Offspring population must be sorted.

### Warning

The function does not break ties randomly as it should.

Definition at line 93 of file population.cc.

```
5.73.2.2 get_best_bv() [1/2]
```

Get best bit vector.

#### **Parameters**

*i* Index in the sorted population

### Precondition

The population must be sorted.

Definition at line 90 of file population.hh.

```
5.73.2.3 get_best_bv() [2/2]
```

```
const bit_vector_t& get_best_bv ( ) const [inline]
```

Get best bit vector.

### Precondition

The population must be sorted.

Definition at line 96 of file population.hh.

```
5.73.2.4 get_best_value() [1/2]
```

```
double get_best_value ( \quad \text{ int } i \text{ ) const [inline]}
```

Get best value.

### **Parameters**

*i* Index in the sorted population

### Precondition

The population must be sorted.

Definition at line 119 of file population.hh.

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

Get best value.

### Precondition

The population must be sorted.

Definition at line 125 of file population.hh.

### 5.73.2.6 get\_worst\_bv()

Get worst bit vector.

#### **Parameters**

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

## Precondition

The population must be sorted.

Definition at line 104 of file population.hh.

## 5.73.2.7 plus\_selection()

Plus selection.

### Precondition

Both populations must be sorted.

### Warning

The function does not break ties randomly as it should.

Definition at line 74 of file population.cc.

## 5.73.3 Member Data Documentation

### 5.73.3.1 \_compare\_index\_value

```
std::function<bool(const index_value_t&, const index_value_t&)> _compare_index_value [protected]
```

### Initial value:

```
[](const index_value_t& a, const index_value_t& b) { return a.second > b.
    second; }
```

Binary operator for comparing index-value pairs.

Definition at line 57 of file population.hh.

## 5.73.3.2 \_lookup

```
std::vector<index_value_t> _lookup [protected]
```

Lookup table.

Let p be of type std::pair<size\_t, double>. Then p.first is the bv index in the unsorted population whereas p.second is the bv value.

Definition at line 54 of file population.hh.

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

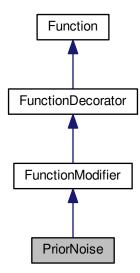
- · lib/hnco/algorithms/population.hh
- lib/hnco/algorithms/population.cc

## 5.74 PriorNoise Class Reference

Prior noise.

#include <hnco/functions/decorators/prior-noise.hh>

Inheritance diagram for PriorNoise:



## **Public Member Functions**

PriorNoise (Function \*fn, neighborhood::Neighborhood \*nh)
 Constructor.

## Information about the function

• size\_t get\_bv\_size ()

Get bit vector size.

• double get\_maximum ()

Get the global maximum.

• bool has\_known\_maximum ()

Check for a known maximum.

• bool provides\_incremental\_evaluation ()

Check whether the function provides incremental evaluation.

### **Evaluation**

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

## **Private Attributes**

neighborhood::Neighborhood \* \_neighborhood
 Neighborhood.

bit\_vector\_t \_noisy\_bv

Noisy bit vector.

## **Additional Inherited Members**

## 5.74.1 Detailed Description

Prior noise.

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

## 5.74.2 Member Function Documentation

```
5.74.2.1 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.

### 5.74.2.2 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.

### 5.74.2.3 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:

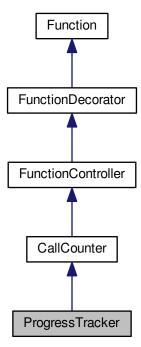
- · lib/hnco/functions/decorators/prior-noise.hh
- lib/hnco/functions/decorators/prior-noise.cc

# 5.75 ProgressTracker Class Reference

### ProgressTracker.

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

Inheritance diagram for ProgressTracker:



### Classes

struct Event

Event.

### **Public Member Functions**

ProgressTracker (Function \*function)

Constructor.

const Event & get\_last\_improvement ()

Get the last improvement.

#### **Evaluation**

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

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

Incremental evaluation.

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

Update after a safe evaluation.

### Setters

void set\_log\_improvement (bool x)

Log improvement.

void set\_stream (std::ostream \*x)

Output stream.

### **Protected Member Functions**

void update\_last\_improvement (double value)

Update last improvement.

### **Protected Attributes**

• Event \_last\_improvement

Last improvement.

### **Parameters**

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

Log improvement.

std::ostream \* \_stream = &std::cout

Output stream.

## 5.75.1 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.

### 5.75.2 Member Function Documentation

Evaluate a bit vector.

### **Exceptions**

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

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

### 5.75.2.2 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\_\(\lefta\) improvement has therefore no meaning.

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

### 5.75.2.3 incremental\_eval()

Incremental evaluation.

#### **Exceptions**

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

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

#### 5.75.2.4 update()

Update after a safe evaluation.

### **Exceptions**

MaximumReached	
TargetReached	

Reimplemented from CallCounter.

Definition at line 191 of file function-controller.cc.

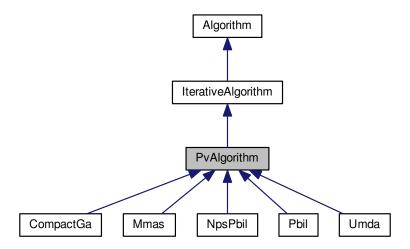
The documentation for this class was generated from the following files:

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

# 5.76 PvAlgorithm Class Reference

Probability vector algorithm.

```
#include <hnco/algorithms/pv/pv-algorithm.hh>
Inheritance diagram for PvAlgorithm:
```



# **Public Types**

- enum { LOG PV, LOG ENTROPY, LAST\_LOG }
- typedef std::bitset< LAST\_LOG > log\_flags\_t

#### **Public Member Functions**

• PvAlgorithm (int n)

Constructor.

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

Set log flags.

#### **Setters**

void set\_log\_num\_components (int x)
 Set the number of probability vector components to log.

#### **Protected Member Functions**

void log ()
 Log.

### **Protected Attributes**

pv\_t \_pv

Probability vector.

• double \_lower\_bound

Lower bound of probability.

double <u>upper\_bound</u>

Upper bound of probability.

• log\_flags\_t \_log\_flags Log flags.

### **Parameters**

• int \_log\_num\_components = 5

Number of probability vector components to log.

# 5.76.1 Detailed Description

Probability vector algorithm.

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

#### 5.76.2 Member Enumeration Documentation

# 5.76.2.1 anonymous enum

anonymous enum

#### Enumerator

LOG_PV	Log probability vector.
LOG_ENTROPY	Log entropy.

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

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

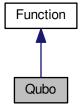
- lib/hnco/algorithms/pv/pv-algorithm.hh
- lib/hnco/algorithms/pv/pv-algorithm.cc

# 5.77 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.

5.77 Qubo Class Reference 167

### 5.77.1 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.

# 5.77.2 Member Function Documentation

```
5.77.2.1 load()
```

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

Load an instance.

Exceptions

Error

Definition at line 35 of file qubo.cc.

# 5.77.3 Member Data Documentation

```
5.77.3.1 _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

# 5.78 Random Struct Reference

#### Random numbers.

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

### **Static Public Member Functions**

• static double uniform ()

Next uniformly distributed sample.

• static double normal ()

Next normally distributed sample.

• static bool random\_bit ()

Next random bit.

### **Static Public Attributes**

 static std::mt19937 engine Engine.

### 5.78.1 Detailed Description

Random numbers.

Definition at line 33 of file random.hh.

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

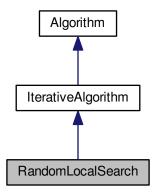
- · lib/hnco/random.hh
- · lib/hnco/random.cc

# 5.79 RandomLocalSearch Class Reference

Random local search.

#include <hnco/algorithms/ls/random-local-search.hh>

Inheritance diagram for RandomLocalSearch:



### **Public Member Functions**

• RandomLocalSearch (int n, neighborhood::Neighborhood \*neighborhood)

Constructor.

· void init ()

Random initialization.

void init (const bit\_vector\_t &x)

Explicit initialization.

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

Explicit initialization.

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

Solution.

#### **Setters**

void set\_compare (std::function< bool(double, double)> x)

Set the binary operator for comparing evaluations.

• void set\_patience (int x)

Set patience.

• void set\_incremental\_evaluation (bool x)

Set incremental evaluation.

### **Protected Member Functions**

```
• void iterate ()
```

Single iteration.

• void iterate\_full ()

Single iteration with full evaluation.

• void iterate\_incremental ()

Single iteration with incremental evaluation.

### **Protected Attributes**

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

Neighborhood.

• int \_num\_failures

Number of failure.

#### **Parameters**

- std::function< bool(double, double)> \_compare = std::greater\_equal<double>()
   Binary operator for comparing evaluations.
- int \_patience = 50

Patience.

• bool \_incremental\_evaluation = false

Incremental evaluation.

# 5.79.1 Detailed Description

Random local search.

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

### 5.79.2 Member Function Documentation

# 5.79.2.1 set\_patience()

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

Set patience.

Number of consecutive rejected moves before throwing a LocalMaximum exception

### Parameters

x Patience

If  $x \le 0$  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:

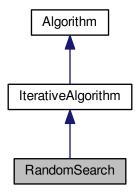
- lib/hnco/algorithms/ls/random-local-search.hh
- lib/hnco/algorithms/ls/random-local-search.cc

# 5.80 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**

# 5.80.1 Detailed Description

Random search.

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

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

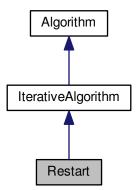
- lib/hnco/algorithms/random-search.hh
- lib/hnco/algorithms/random-search.cc

# 5.81 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**

# 5.81.1 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:

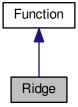
- · lib/hnco/algorithms/restart.hh
- · lib/hnco/algorithms/restart.cc

# 5.82 Ridge Class Reference

# Ridge.

#include <hnco/functions/theory.hh>

Inheritance diagram for Ridge:



# **Public Member Functions**

```
• Ridge (int bv_size)
```

Constructor.

size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

• double get\_maximum ()

Get the global maximum.

### **Private Attributes**

• size\_t \_bv\_size

Bit vector size.

# 5.82.1 Detailed Description

# Ridge.

Reference:

Thomas Jansen. 2013. Analyzing Evolutionary Algorithms. Springer.

Definition at line 203 of file theory.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
```

Reimplemented from Function.

Definition at line 227 of file theory.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 223 of file theory.hh.

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

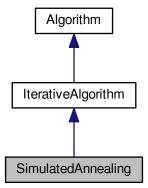
- · lib/hnco/functions/theory.hh
- lib/hnco/functions/theory.cc

# 5.83 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**

### 5.83.1 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.

#### 5.83.2 Member Function Documentation

### 5.83.2.1 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:

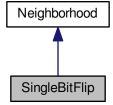
- · lib/hnco/algorithms/ls/simulated-annealing.hh
- · lib/hnco/algorithms/ls/simulated-annealing.cc

# 5.84 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**

# 5.84.1 Detailed Description

One bit neighborhood.

Definition at line 160 of file neighborhood.hh.

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

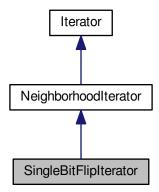
• lib/hnco/neighborhoods/neighborhood.hh

# 5.85 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.

bool has\_next ()

Has next bit vector.

• const bit\_vector\_t & next ()

Next bit vector.

# **Private Attributes**

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

### **Additional Inherited Members**

# 5.85.1 Detailed Description

Single bit flip neighborhood iterator.

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

#### 5.85.2 Constructor & Destructor Documentation

# 5.85.2.1 SingleBitFlipIterator()

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

Constructor.

# Parameters

n Size of bit vectors

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

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

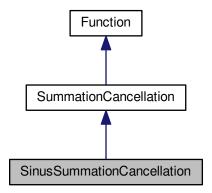
- lib/hnco/neighborhoods/neighborhood-iterator.hh
- lib/hnco/neighborhoods/neighborhood-iterator.cc

# 5.86 SinusSummationCancellation Class Reference

Summation cancellation with sinus.

#include <hnco/functions/cancellation.hh>

Inheritance diagram for SinusSummationCancellation:



# **Public Member Functions**

• SinusSummationCancellation (int n)

Constructor.

• double eval (const bit\_vector\_t &x)

Evaluate a bit vector.

#### **Additional Inherited Members**

### 5.86.1 Detailed Description

Summation cancellation with sinus.

### Reference:

M. Sebag and M. Schoenauer. 1997. A society of hill-climbers. In Proc. IEEE Int. Conf. on Evolutionary Computation. Indianapolis, 319–324.

Definition at line 103 of file cancellation.hh.

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

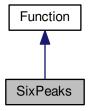
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

# 5.87 SixPeaks Class Reference

Six Peaks.

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

Inheritance diagram for SixPeaks:



# **Public Member Functions**

• SixPeaks (int bv\_size, int threshold)

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

• double eval (const bit\_vector\_t &)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

• double get\_maximum ()

Get the global maximum.

# **Private Attributes**

size\_t \_bv\_size

Bit vector size.

• int \_threshold

Threshold.

• int \_maximum

Maximum.

# 5.87.1 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(1111110) = 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.

#### 5.87.2 Member Function Documentation

```
5.87.2.1 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.

```
5.87.2.2 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

# 5.88 SpinHerding Class Reference

Herding with spin variables.

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

# **Public Types**

enum {
 SAMPLE\_GREEDY, SAMPLE\_RLS, SAMPLE\_DLS, SAMPLE\_NN,
 LAST\_SAMPLE }

# **Public Member Functions**

• SpinHerding (int n)

Constructor.

· void init ()

Initialization.

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

Sample a bit vector.

• double error (const SpinMoment &target)

Compute the error.

• double delta (const SpinMoment &target)

Compute the norm of the moment increment.

#### **Setters**

void set randomize bit order (bool x)

Randomize bit order.

void set\_sampling\_method (int x)

Set the sampling method.

void set\_num\_seq\_updates (int x)

Set the number of sequential updates per sample.

void set\_num\_par\_updates (int x)

Set the number of parallel updates per sample.

void set\_weight (double x)

Set the weight of second order moments.

#### **Protected Member Functions**

· void compute\_delta (const SpinMoment &target)

Compute delta.

void sample\_greedy (bit\_vector\_t &x)

Sample by means of a greedy algorithm.

double q\_derivative (const bit\_vector\_t &x, size\_t i)

Derivative of q.

double q\_variation (const bit\_vector\_t &x, size\_t i)

Variation of q.

void sample\_rls (bit\_vector\_t &x)

Sample by means of random local search.

void sample\_dls (bit\_vector\_t &x)

Sample by means of deterministic local search.

void sample\_nn (bit\_vector\_t &x)

Sample by means of a neural network.

void update\_counters (const bit\_vector\_t &x)

Update counters.

#### **Protected Attributes**

· SpinMoment \_delta

Delta moment.

SpinMoment \_count

Counter moment.

bit\_vector\_t \_state

State.

permutation\_t \_permutation

Permutation.

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

Choose bit.

· int \_time

Time.

#### **Parameters**

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

Randomize bit order.

int \_sampling\_method = SAMPLE\_GREEDY

Sampling method.

· int num seq updates

Number of sequential updates per sample.

• int \_num\_par\_updates = 1

Number of parallel updates per sample.

• double \_weight = 1

Weight of second order moments.

### 5.88.1 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.

### 5.88.2 Member Enumeration Documentation

# 5.88.2.1 anonymous enum

anonymous enum

#### Enumerator

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

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

# 5.88.3 Constructor & Destructor Documentation

# 5.88.3.1 SpinHerding()

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

Constructor.

#### **Parameters**

```
n Size of bit vectors
```

\_num\_seq\_updates is initialized to n.

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

#### 5.88.4 Member Function Documentation

# 5.88.4.1 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

# 5.89 SpinMoment Struct Reference

Moment for spin variables.

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

#### **Public Member Functions**

• SpinMoment (int n)

Constructor.

· void uniform ()

Set the moment to that of the uniform distribution.

· void init ()

Initialize accumulators.

void add (const bit\_vector\_t &x)

Update accumulators.

void average (int count)

Compute average.

• void update (const SpinMoment &p, double rate)

Update moment.

void bound (double margin)

Bound moment.

· double distance (const SpinMoment &p) const

Distance.

• double norm\_2 () const

Compute the norm 2.

• double diameter () const

Compute the diameter.

· size\_t size () const

Size.

#### **Public Attributes**

std::vector< double > \_first

First moment.

std::vector< std::vector< double >> \_second

Second moment.

• double \_weight = 1

Weight of second order moments.

# 5.89.1 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:

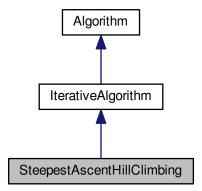
- lib/hnco/algorithms/hea/moment-spin.hh
- lib/hnco/algorithms/hea/moment-spin.cc

# 5.90 SteepestAscentHillClimbing Class Reference

Steepest ascent hill climbing.

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

Inheritance diagram for SteepestAscentHillClimbing:



### **Public Member Functions**

• SteepestAscentHillClimbing (int n, neighborhood::NeighborhoodIterator \*neighborhood)

Constructor.

· void init ()

Random initialization.

void init (const bit\_vector\_t &x)

Explicit initialization.

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

Explicit initialization.

# **Protected Member Functions**

• void iterate ()

Single iteration.

# **Protected Attributes**

std::vector < bit\_vector\_t > \_candidates
 Potential candidate.

neighborhood::NeighborhoodIterator \* \_neighborhood
 Neighborhood.

# 5.90.1 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:

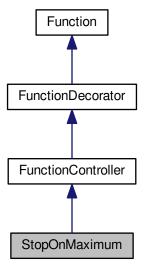
- · lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.hh
- lib/hnco/algorithms/ls/steepest-ascent-hill-climbing.cc

# 5.91 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**

# 5.91.1 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.

#### 5.91.2 Constructor & Destructor Documentation

### 5.91.2.1 StopOnMaximum()

Constructor.

**Parameters** 

```
function Decorated function
```

#### Precondition

function->has\_known\_maximum()

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

# 5.91.3 Member Function Documentation

Evaluate a bit vector.

**Exceptions** 

MaximumReached

Implements Function.

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

### 5.91.3.2 incremental\_eval()

Incremental evaluation.

**Exceptions** 

MaximumReached

Reimplemented from Function.

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

#### 5.91.3.3 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:

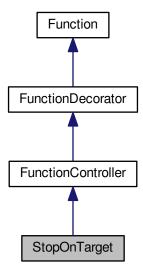
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 5.92 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**

# 5.92.1 Detailed Description

Stop on target.

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

#### 5.92.2 Constructor & Destructor Documentation

### 5.92.2.1 StopOnTarget()

Constructor.

**Parameters** 

```
function Decorated function
```

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

# 5.92.3 Member Function Documentation

Evaluate a bit vector.

**Exceptions** 

TargetReached

Implements Function.

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

#### 5.92.3.2 incremental\_eval()

Incremental evaluation.

**Exceptions** 

**TargetReached** 

Reimplemented from Function.

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

#### 5.92.3.3 update()

Update after a safe evaluation.

**Exceptions** 

TargetReached

Reimplemented from Function.

Definition at line 86 of file function-controller.cc.

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

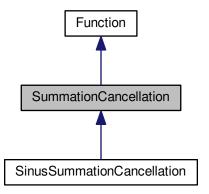
- · lib/hnco/functions/decorators/function-controller.hh
- lib/hnco/functions/decorators/function-controller.cc

# 5.93 SummationCancellation Class Reference

Summation cancellation.

#include <hnco/functions/cancellation.hh>

Inheritance diagram for SummationCancellation:



#### **Public Member Functions**

• SummationCancellation (int n)

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

double eval (const bit\_vector\_t &x)

Evaluate a bit vector.

• bool has\_known\_maximum ()

Check for a known maximum.

• double get\_maximum ()

Get the global maximum.

### **Protected Member Functions**

void convert (const bit\_vector\_t &x)
 Convert a bit vector into a real vector.

# **Protected Attributes**

• size\_t \_bv\_size

Bit vector size.

•  $std::vector < double > \_buffer$ 

Buffer.

# 5.93.1 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.

### 5.93.2 Constructor & Destructor Documentation

#### 5.93.2.1 SummationCancellation()

```
\label{eq:continuous} \begin{tabular}{ll} Summation Cancellation ( \\ & int \ n \ ) & [inline] \end{tabular}
```

Constructor.

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

#### **Parameters**

```
n Size of the bit vector
```

Definition at line 70 of file cancellation.hh.

#### 5.93.3 Member Function Documentation

# 5.93.3.1 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:

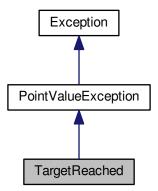
- · lib/hnco/functions/cancellation.hh
- lib/hnco/functions/cancellation.cc

# 5.94 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**

# 5.94.1 Detailed Description

target reached

Definition at line 61 of file exception.hh.

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

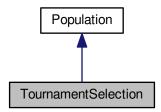
· lib/hnco/exception.hh

# 5.95 TournamentSelection Class Reference

Population with tournament selection.

#include <hnco/algorithms/ea/tournament-selection.hh>

Inheritance diagram for TournamentSelection:



### **Public Member Functions**

• TournamentSelection (int population\_size, int n)

Constructor.

• const bit\_vector\_t & select () Selection.

### Setters

void set\_tournament\_size (int x)
 Set the tournament size.

### **Private Attributes**

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

### **Parameters**

• int \_tournament\_size = 10 Tournament size.

### **Additional Inherited Members**

# 5.95.1 Detailed Description

Population with tournament selection.

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

# 5.95.2 Member Function Documentation

# 5.95.2.1 select()

```
const bit_vector_t & select ( )
```

Selection.

The selection only requires that the population be evaluated, not necessarily sorted.

#### Precondition

The population must be evaluated.

Definition at line 33 of file tournament-selection.cc.

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

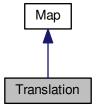
- lib/hnco/algorithms/ea/tournament-selection.hh
- lib/hnco/algorithms/ea/tournament-selection.cc

# 5.96 Translation Class Reference

# Translation.

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

Inheritance diagram for Translation:



#### **Public Member Functions**

```
    void random (int n)
        Random instance.

    void map (const bit_vector_t &input, bit_vector_t &output)
        Map.
```

size\_t get\_input\_size ()

Get input size.

• size\_t get\_output\_size ()

Get output size.

• bool is\_surjective ()

Check for surjective map.

## **Private Member Functions**

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

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

#### **Private Attributes**

bit\_vector\_t \_bv
 Translation vector.

## **Friends**

· class boost::serialization::access

## 5.96.1 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.

#### 5.96.2 Member Function Documentation

#### 5.96.2.1 is\_surjective()

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

Check for surjective map.

Returns

true

Reimplemented from Map.

Definition at line 121 of file map.hh.

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

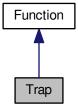
- · lib/hnco/map.hh
- · lib/hnco/map.cc

## 5.97 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.

## 5.97.1 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.

#### 5.97.2 Constructor & Destructor Documentation

int num\_traps ) [inline]

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.

#### 5.97.3 Member Function Documentation

# 5.97.3.1 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. 5.97.3.2 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

## 5.98 Umda Class Reference

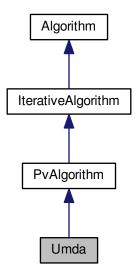
Univariate marginal distribution algorithm.

#include <hnco/algorithms/pv/umda.hh>

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

5.98 Umda Class Reference 203

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**

## 5.98.1 Detailed Description

Univariate marginal distribution algorithm.

Reference:

H. Mühlenbein. 1997. The equation for response to selection and its use for prediction. Evolutionary Computation 5, 3 (1997), 303–346.

Definition at line 40 of file umda.hh.

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

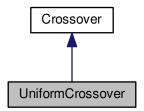
- · lib/hnco/algorithms/pv/umda.hh
- · lib/hnco/algorithms/pv/umda.cc

## 5.99 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.

## 5.99.1 Detailed Description

Uniform crossover.

Definition at line 56 of file crossover.hh.

#### 5.99.2 Member Function Documentation

#### 5.99.2.1 breed()

Breed.

The offspring is the uniform crossover of two parents.

#### **Parameters**

parent1	First parent
parent2	Second parent
offspring	Offspring

Implements Crossover.

Definition at line 30 of file crossover.cc.

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

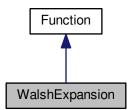
- · lib/hnco/algorithms/ea/crossover.hh
- lib/hnco/algorithms/ea/crossover.cc

## 5.100 WalshExpansion Class Reference

Walsh expansion.

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

Inheritance diagram for WalshExpansion:



#### **Public Member Functions**

```
· WalshExpansion ()
```

Constructor.

size\_t get\_bv\_size ()

Get bit vector size.

· void random (int n, int num features, double stddev)

Random instance.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

void display (std::ostream &stream)

Display

void set terms (const std::vector< Function::WalshTransformTerm > terms)

Set terms.

#### **Private Member Functions**

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

## **Private Attributes**

std::vector< Function::WalshTransformTerm > \_terms
 Terms.

#### **Friends**

· class boost::serialization::access

## 5.100.1 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.

#### 5.100.2 Member Function Documentation

## 5.100.2.1 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:

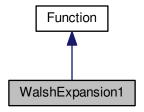
- · lib/hnco/functions/walsh/walsh-expansion.hh
- lib/hnco/functions/walsh/walsh-expansion.cc

## 5.101 WalshExpansion1 Class Reference

Walsh expansion of degree 1.

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

Inheritance diagram for WalshExpansion1:



## **Public Member Functions**

• WalshExpansion1 ()

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

• void random (int n, double stddev)

Random instance.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

## **Private Member Functions**

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

## **Private Attributes**

std::vector< double > \_linear
 Linear part.

## **Friends**

· class boost::serialization::access

## 5.101.1 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.

## 5.101.2 Member Function Documentation

#### 5.101.2.1 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:

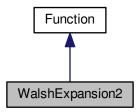
- · lib/hnco/functions/walsh/walsh-expansion-1.hh
- lib/hnco/functions/walsh/walsh-expansion-1.cc

## 5.102 WalshExpansion2 Class Reference

Walsh expansion of degree 2.

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

Inheritance diagram for WalshExpansion2:



## **Public Member Functions**

• WalshExpansion2 ()

Constructor.

• size\_t get\_bv\_size ()

Get bit vector size.

void random (int n, double stddev\_lin, double stddev\_quad)

Random instance.

double eval (const bit\_vector\_t &)

Evaluate a bit vector.

#### **Private Member Functions**

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

## **Private Attributes**

std::vector< double > \_linear

Linear part.

• std::vector< std::vector< double >> \_quadratic

Quadratic part.

## Friends

· class boost::serialization::access

## 5.102.1 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.

#### 5.102.2 Member Function Documentation

#### 5.102.2.1 random()

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.

#### 5.102.3 Member Data Documentation

## 5.102.3.1 \_quadratic

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

Quadratic part.

Represented as a lower triangular matrix (without its diagonal).

Definition at line 75 of file walsh-expansion-2.hh.

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

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

## 5.103 Function::WalshTransformTerm Struct Reference

Walsh transform term.

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

#### **Public Member Functions**

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

#### **Public Attributes**

- std::vector< bool > feature
   Feature.
- · double coefficient

Coefficient.

#### 5.103.1 Detailed Description

Walsh transform term.

Definition at line 44 of file function.hh.

#### 5.103.2 Member Data Documentation

## 5.103.2.1 feature

```
std::vector<bool> feature
```

#### Feature.

Implemented with a vector bool instead of a bit\_vector\_t to reduce the memory consumption.

Definition at line 51 of file function.hh.

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

· lib/hnco/functions/function.hh

## Index

_compare_index_value	hnco, 19
hnco::algorithm::Population, 158	Cache, 44
_expression	hnco::function::Cache, 45
hnco::function::MaxSat, 114	CallCounter, 46
_functions	
hnco::algorithm::Algorithm, 33	comma_selection
_lookup	hnco::algorithm::Population, 155
hnco::algorithm::Population, 158	CompactGa, 47
_q	CompleteSearch, 49
hnco::function::Qubo, 168	compute_walsh_transform
_quadratic	hnco::function::Function, 64
hnco::function::WalshExpansion2, 210	Crossover, 50
AdditiveGaussianNoise, 27	DeceptiveJump, 51
AffineMap, 29	FavralDraduata F0
Algorithm, 31	EqualProducts, 53
•	Error, 55
bernoulli trials	eval
hnco::neighborhood::MultiBitFlip, 121	hnco::function::OnBudgetFunction, 139
BernoulliProcess, 33	hnco::function::ProgressTracker, 163
hnco::neighborhood::BernoulliProcess, 34, 35	hnco::function::StopOnMaximum, 190
BiasedCrossover, 36	hnco::function::StopOnTarget, 192
BinaryHerding, 37	Exception, 56
BinaryMoment, 39	Factorization E7
bit_t	Factorization, 57
hnco, 14	hnco::function::Factorization, 58
bm_add_rows	feature
hnco, 15	hnco::function::Function::WalshTransformTerm
bm_identity	211
hnco, 15	FirstAscentHillClimbing, 59
bm invert	FourPeaks, 60
hnco, 16	Function, 63
bm multiply	Function::WalshTransformTerm, 211
_ , ,	FunctionController, 67
hnco, 16	FunctionDecorator, 68
bm_rank	FunctionMapComposition, 69
hnco, 17	hnco::function::FunctionMapComposition, 70
bm_row_echelon_form	FunctionModifier, 72
hnco, 17	FunctionPlugin, 73
bm_solve	hnco::function::FunctionPlugin, 74
hnco, 17	
bm_solve_upper_triangular	GeneticAlgorithm, 74
hnco, 18	hnco::algorithm::GeneticAlgorithm, 76
BmPbil, 40	get_best_bv
breed	hnco::algorithm::Population, 155, 156
hnco::algorithm::BiasedCrossover, 36	get_best_value
hnco::algorithm::Crossover, 51	hnco::algorithm::Population, 156
hnco::algorithm::UniformCrossover, 205	get_last_improvement
bv_from_vector_bool	hnco::function::ProgressTracker, 163
hnco, 19	get_maximum
bv_to_vector_bool	hnco::function::AdditiveGaussianNoise, 28

hnco::function::DeceptiveJump, 52	sbm_multiply, 19
hnco::function::FourPeaks, 62	sparse_bit_matrix_t, 14
hnco::function::Function, 65	sparse_bit_vector_t, 15
hnco::function::FunctionMapComposition, 71	hnco::AffineMap
hnco::function::Hiff, 89	is_surjective, 30
hnco::function::Jump, 97	random, 30
hnco::function::LeadingOnes, 100	hnco::LinearMap
hnco::function::Needle, 126	•
	is_surjective, 104
hnco::function::Negation, 128	random, 104
hnco::function::OneMax, 141	hnco::Map
hnco::function::Plateau, 152	is_surjective, 109
hnco::function::PriorNoise, 160	hnco::MapComposition
hnco::function::Ridge, 174	is_surjective, 111
hnco::function::SixPeaks, 182	MapComposition, 110
hnco::function::Trap, 201	hnco::Permutation
get_worst_bv	is_surjective, 150
hnco::algorithm::Population, 157	hnco::Translation
missaigsmini opaidism, 107	
HammingBall, 77	is_surjective, 199
hnco::neighborhood::HammingBall, 78	hnco::algorithm, 20
HammingSphere, 79	hnco::algorithm::Algorithm
<del>-</del> •	_functions, 33
hnco::neighborhood::HammingSphere, 80	hnco::algorithm::BiasedCrossover
HammingSphereIterator, 80	breed, 36
hnco::neighborhood::HammingSphereIterator, 82	hnco::algorithm::Crossover
has_known_maximum	breed, 51
hnco::function::AdditiveGaussianNoise, 28	
hnco::function::DeceptiveJump, 52	hnco::algorithm::GeneticAlgorithm
hnco::function::FourPeaks, 62	GeneticAlgorithm, 76
hnco::function::FunctionMapComposition, 71	set_allow_stay, 77
hnco::function::Hiff, 89	hnco::algorithm::IterativeAlgorithm
hnco::function::Jump, 97	IterativeAlgorithm, 93
hnco::function::LeadingOnes, 100	maximize, 93
<del></del>	set_num_iterations, 94
hnco::function::LinearFunction, 102	hnco::algorithm::MuCommaLambdaEa
hnco::function::Needle, 126	MuCommaLambdaEa, 119
hnco::function::Negation, 128	set_allow_stay, 120
hnco::function::OneMax, 141	_ · ·
hnco::function::Plateau, 152	hnco::algorithm::MuPlusLambdaEa
hnco::function::PriorNoise, 160	MuPlusLambdaEa, 124
hnco::function::Ridge, 174	set_allow_stay, 124
hnco::function::SixPeaks, 182	hnco::algorithm::OnePlusLambdaCommaLambdaGa
hnco::function::SummationCancellation, 195	OnePlusLambdaCommaLambdaGa, 144
hnco::function::Trap, 202	hnco::algorithm::OnePlusOneEa
Hboa, 82	OnePlusOneEa, 146
Hea	set_allow_stay, 146
hnco::algorithm::hea::Hea, 87	set num iterations, 146
_	hnco::algorithm::Population
Hea< Moment, Herding >, 83	•
Hiff, 88	_compare_index_value, 158
hnco, 11	_lookup, 158
bit_t, 14	comma_selection, 155
bm_add_rows, 15	get_best_bv, 155, 156
bm_identity, 15	get_best_value, 156
bm_invert, 16	get_worst_bv, 157
bm_multiply, 16	plus_selection, 157
bm_rank, 17	hnco::algorithm::RandomLocalSearch
bm_row_echelon_form, 17	set_patience, 170
bm_solve, 17	hnco::algorithm::SimulatedAnnealing
	-
bm_solve_upper_triangular, 18	init_beta, 177
bv_from_vector_bool, 19	hnco::algorithm::TournamentSelection
bv_to_vector_bool, 19	select, 198

hnco::algorithm::UniformCrossover	random, 102
breed, 205	hnco::function::MaxSat
hnco::algorithm::bm_pbil, 22	_expression, 114
hnco::algorithm::bm_pbil::BmPbil	load, 113
set_selection_size, 43	random, 113, 114
hnco::algorithm::hea, 22	hnco::function::Needle
hnco::algorithm::hea::Hea	get_maximum, 126
Hea, 87	has_known_maximum, 126
set_reset_period, 87	hnco::function::Negation
set_selection_size, 87	get_maximum, 128
hnco::algorithm::hea::SpinHerding	has_known_maximum, 128
q_variation, 185	provides_incremental_evaluation, 128
SpinHerding, 185	hnco::function::NkLandscape
hnco::exception, 23	random, 135
hnco::function, 23	hnco::function::OnBudgetFunction
hnco::function::AdditiveGaussianNoise	eval, 139
get_maximum, 28	incremental_eval, 139
has_known_maximum, 28	update, 139
hnco::function::Cache	hnco::function::OneMax
Cache, 45	get_maximum, 141
provides incremental evaluation, 45	has_known_maximum, 141
hnco::function::DeceptiveJump	provides_incremental_evaluation, 142
get maximum, 52	hnco::function::Plateau
has_known_maximum, 52	get_maximum, 152
hnco::function::EqualProducts	has_known_maximum, 152
random, 54	hnco::function::PriorNoise
hnco::function::Factorization	get_maximum, 160
Factorization, 58	has_known_maximum, 160
hnco::function::FourPeaks	provides_incremental_evaluation, 160
get_maximum, 62	hnco::function::ProgressTracker
has_known_maximum, 62	eval, 163
hnco::function::Function	get_last_improvement, 163
compute_walsh_transform, 64	incremental_eval, 163
get_maximum, 65	update, 164
incremental_eval, 65	hnco::function::Qubo
provides_incremental_evaluation, 66	_q, 168
safe_eval, 66	load, 167
hnco::function::WalshTransformTerm	hnco::function::Ridge
feature, 211	get_maximum, 174
hnco::function::FunctionController	has known maximum, 174
provides incremental evaluation, 68	hnco::function::SixPeaks
hnco::function::FunctionMapComposition	get_maximum, 182
FunctionMapComposition, 70	has_known_maximum, 182
get maximum, 71	hnco::function::StopOnMaximum
has_known_maximum, 71	eval, 190
hnco::function::FunctionPlugin	incremental_eval, 190
FunctionPlugin, 74	StopOnMaximum, 189
hnco::function::Hiff	update, 190
get_maximum, 89	hnco::function::StopOnTarget
has_known_maximum, 89	eval, 192
hnco::function::Jump	incremental_eval, 193
get_maximum, 97	StopOnTarget, 192
has_known_maximum, 97	update, 193
hnco::function::LeadingOnes	hnco::function::SummationCancellation
get_maximum, 100	has_known_maximum, 195
has_known_maximum, 100	SummationCancellation, 195
hnco::function::LinearFunction	hnco::function::Trap
has known maximum, 102	get_maximum, 201
	300

has_known_maximum, 202	LeadingOnes, 99
Trap, 201	LinearFunction, 101
hnco::function::WalshExpansion	LinearMap, 103
random, 206	load
hnco::function::WalshExpansion1	hnco::function::MaxSat, 113
random, 208	hnco::function::Qubo, 167
hnco::function::WalshExpansion2	LocalMaximum, 105
_quadratic, 210	LongPath, 106
random, 210	Ltga, 107
hnco::neighborhood, 25	
hnco::neighborhood::BernoulliProcess	Map, 108
BernoulliProcess, 34, 35	map
set_allow_stay, 35	hnco::neighborhood::Neighborhood, 131
set_probability, 35	MapComposition, 109
hnco::neighborhood::HammingBall	hnco::MapComposition, 110
HammingBall, 78	MaxSat, 112
hnco::neighborhood::HammingSphere	maximize
HammingSphere, 80	hnco::algorithm::IterativeAlgorithm, 93
hnco::neighborhood::HammingSphereIterator	MaximumReached, 111
HammingSphereIterator, 82	Mmas, 115
hnco::neighborhood::MultiBitFlip	Model, 116
bernoulli trials, 121	ModelParameters, 117
MultiBitFlip, 121	MuCommaLambdaEa, 118
reservoir sampling, 122	hnco::algorithm::MuCommaLambdaEa, 119
hnco::neighborhood::Neighborhood	MuPlusLambdaEa, 122
map, 131	hnco::algorithm::MuPlusLambdaEa, 124
mutate, 132	MultiBitFlip, 120
Neighborhood, 131	hnco::neighborhood::MultiBitFlip, 121
hnco::neighborhood::NeighborhoodIterator	mutate
NeighborhoodIterator, 133	hnco::neighborhood::Neighborhood, 132
hnco::neighborhood::SingleBitFlipIterator	
SingleBitFlipIterator, 179	Needle, 125
hnco::random, 26	Negation, 127
HncoEvaluator, 90	Neighborhood, 129
Hypercubelterator, 91	hnco::neighborhood::Neighborhood, 131
Tiyporoabonorator, or	NeighborhoodIterator, 132
incremental eval	hnco::neighborhood::NeighborhoodIterator, 133
hnco::function::Function, 65	NkLandscape, 133
hnco::function::OnBudgetFunction, 139	NpsPbil, 135
hnco::function::ProgressTracker, 163	•
hnco::function::StopOnMaximum, 190	OnBudgetFunction, 137
hnco::function::StopOnTarget, 193	OneMax, 140
init_beta	OnePlusLambdaCommaLambdaGa, 142
hnco::algorithm::SimulatedAnnealing, 177	hnco::algorithm::OnePlusLambdaComma←
is_surjective	LambdaGa, 144
hnco::AffineMap, 30	OnePlusOneEa, 144
hnco::LinearMap, 104	hnco::algorithm::OnePlusOneEa, 146
hnco::Map, 109	
hnco::MapComposition, 111	ParameterlessPopulationPyramid, 147
hnco::Permutation, 150	Pbil, 148
hnco::Translation, 199	Permutation, 149
IterativeAlgorithm, 92	Plateau, 151
hnco::algorithm::IterativeAlgorithm, 93	plus_selection
Iterator, 94	hnco::algorithm::Population, 157
•	PointValueException, 153
Jump, 96	Population, 154
	PriorNoise, 159
Labs, 97	ProgressTracker, 161
LastEvaluation, 98	ProgressTracker::Event, 56

provides_incremental_evaluation	SinusSummationCancellation, 180
hnco::function::Cache, 45	SixPeaks, 181
hnco::function::Function, 66	sparse_bit_matrix_t
hnco::function::FunctionController, 68	hnco, 14
hnco::function::Negation, 128	sparse_bit_vector_t
hnco::function::OneMax, 142	hnco, 15
hnco::function::PriorNoise, 160	SpinHerding, 183
PvAlgorithm, 164	hnco::algorithm::hea::SpinHerding, 185
	SpinMoment, 186
q_variation	SteepestAscentHillClimbing, 187
hnco::algorithm::hea::SpinHerding, 185	StopOnMaximum, 188
Qubo, 166	hnco::function::StopOnMaximum, 189
	StopOnTarget, 191
Random, 168	hnco::function::StopOnTarget, 192
random	SummationCancellation, 193
hnco::AffineMap, 30	hnco::function::SummationCancellation, 195
hnco::LinearMap, 104	
hnco::function::EqualProducts, 54	TargetReached, 196
hnco::function::LinearFunction, 102	TournamentSelection, 197
hnco::function::MaxSat, 113, 114	Translation, 198
hnco::function::NkLandscape, 135	Trap, 200
hnco::function::WalshExpansion, 206	hnco::function::Trap, 201
hnco::function::WalshExpansion1, 208	
hnco::function::WalshExpansion2, 210	Umda, 202
RandomLocalSearch, 169	UniformCrossover, 204
RandomSearch, 171	update
reservoir_sampling	hnco::function::OnBudgetFunction, 139
hnco::neighborhood::MultiBitFlip, 122	hnco::function::ProgressTracker, 164
Restart, 172	hnco::function::StopOnMaximum, 190
Ridge, 173	hnco::function::StopOnTarget, 193
Ridge, 173	•
Ridge, 173 safe_eval	WalshExpansion, 205
	WalshExpansion, 205 WalshExpansion1, 207
safe_eval	WalshExpansion, 205
safe_eval hnco::function::Function, 66	WalshExpansion, 205 WalshExpansion1, 207
safe_eval hnco::function::Function, 66 sbm_multiply	WalshExpansion, 205 WalshExpansion1, 207
safe_eval hnco::function::Function, 66 sbm_multiply hnco, 19	WalshExpansion, 205 WalshExpansion1, 207
safe_eval hnco::function::Function, 66 sbm_multiply hnco, 19 select	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66 sbm_multiply     hnco, 19 select     hnco::algorithm::TournamentSelection, 198	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66 sbm_multiply     hnco, 19 select     hnco::algorithm::TournamentSelection, 198 set_allow_stay	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66 sbm_multiply     hnco, 19 select     hnco::algorithm::TournamentSelection, 198 set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66 sbm_multiply     hnco, 19 select     hnco::algorithm::TournamentSelection, 198 set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66 sbm_multiply     hnco, 19 select     hnco::algorithm::TournamentSelection, 198 set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146	WalshExpansion, 205 WalshExpansion1, 207
safe_eval hnco::function::Function, 66  sbm_multiply hnco, 19  select hnco::algorithm::TournamentSelection, 198  set_allow_stay hnco::algorithm::GeneticAlgorithm, 77 hnco::algorithm::MuCommaLambdaEa, 120 hnco::algorithm::MuPlusLambdaEa, 124 hnco::algorithm::OnePlusOneEa, 146 hnco::neighborhood::BernoulliProcess, 35	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::IterativeAlgorithm, 94	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::IterativeAlgorithm, 94     hnco::algorithm::OnePlusOneEa, 146	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::IterativeAlgorithm, 94     hnco::algorithm::OnePlusOneEa, 146  set_patience	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::IterativeAlgorithm, 94     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::IterativeAlgorithm, 94     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35  set_reset_period	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35  set_reset_period     hnco::algorithm::hea::Hea, 87  set_selection_size	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::IterativeAlgorithm, 94     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35  set_reset_period     hnco::algorithm::hea::Hea, 87  set_selection_size     hnco::algorithm::bm_pbil::BmPbil, 43	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35  set_reset_period     hnco::algorithm::hea::Hea, 87  set_selection_size     hnco::algorithm::bm_pbil::BmPbil, 43     hnco::algorithm::bm_pbil::BmPbil, 43     hnco::algorithm::hea::Hea, 87	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::GeneticAlgorithm, 77     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35  set_reset_period     hnco::algorithm::hea::Hea, 87  set_selection_size     hnco::algorithm::bm_pbil::BmPbil, 43     hnco::algorithm::hea::Hea, 87  SimulatedAnnealing, 175	WalshExpansion, 205 WalshExpansion1, 207
safe_eval     hnco::function::Function, 66  sbm_multiply     hnco, 19  select     hnco::algorithm::TournamentSelection, 198  set_allow_stay     hnco::algorithm::MuCommaLambdaEa, 120     hnco::algorithm::MuPlusLambdaEa, 124     hnco::algorithm::OnePlusOneEa, 146     hnco::neighborhood::BernoulliProcess, 35  set_num_iterations     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::OnePlusOneEa, 146  set_patience     hnco::algorithm::RandomLocalSearch, 170  set_probability     hnco::neighborhood::BernoulliProcess, 35  set_reset_period     hnco::algorithm::hea::Hea, 87  set_selection_size     hnco::algorithm::bm_pbil::BmPbil, 43     hnco::algorithm::bm_pbil::BmPbil, 43     hnco::algorithm::hea::Hea, 87	WalshExpansion, 205 WalshExpansion1, 207